Multi-software reliability allocation in multimedia systems with budget constraints using Dempster–Shafer theory and improved differential evolution

Feng Yue a, Guofu Zhang a,b,*, Zhaopin Su a, Yang Lu a,b, Ting Zhang a

*a School of Computer and Information, Hefei University of Technology, Hefei 230009, China
b Engineering Research Center of Safety Critical Industrial Measurement and Control Technology, Ministry of Education, Hefei 230009, China

ARTICLE INFO

Article history:
Received 25 April 2014
Received in revised form 14 July 2014
Accepted 8 September 2014

Keywords:
Multi-software
Reliability allocation
Uncertainty
Dempster–Shafer theory
Differential evolution

ABSTRACT

In multimedia platform with many applications, reliability allocation plays an important role in the design of a software and has attracted increasing attention in recent years. Thus far, the issues of software reliability allocation have been discussed from many aspects, such as mathematical models and solutions to maximize the reliability. However, most of this research has concentrated on single software. The goal of this work is to investigate the possibility of solving multi-software reliability allocation in multimedia systems with budget constraints. For this purpose, we first develop an architecture-based multi-software Budget-Constrained Reliability-maximization model. In addition, we introduce Dempster–Shafer theory to identify the relative reliability weights of each element in the proposed model and present a searching algorithm based on differential evolution and encoding repair. Finally, contrast experiments are illustrated to demonstrate the proposed methods.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

In recent years, the rapid evolution of computing system technologies stimulates research and development in the field of multimedia integrating platform. And many multimedia systems have been deployed with the requirement from sectors such as traffic control, safety critical, education and information management. Failure of these systems can lead to big losses. The critical challenges to software engineers, who need to develop systems with, must assured high reliability levels while at the same time keeping the development time and costs low.

Software reliability is the probability of execution of software without failure under specified environment for a specified period of time [1,2]. In order to achieve the reliability goal of software, many researchers have devoted to various software reliability growth models (SRGMs) [3–6]. However, accurate software reliability estimation is not usually available until the software has been tested by a large number of failure data for a longer period of time. Furthermore, for software engineers, estimating the reliability of a software during the early phase of development is an important requirement for achieving an optimal system reliability goal.

Software reliability allocation can be operated during the design phase of a software, and which is a method of allocating a target reliability among subsystem and components. In the past decades, researchers devoted to constructing architecture-based models for the reliability allocation problem. The idea of software reliability allocation was first put forward by Zahedi and Ashrafi [7], who adopted analytic hierarchy process (AHP) [8] for modeling the software architecture with cost as the constraints and proposed a method for the system reliability maximization. Leung [9] used the operational profile to define a software utility function, which reflects a weighted sum of reliability-like measures based on the same AHP process as [7]. Helander et al. [10] described two approaches for reliability and cost planning: Reliability-Constrained Cost-Minimization (RCCM) and Budget-Constrained Reliability-Maximization (BCRM), both of which are multivariate constrained optimization problems. Rani and Misra [11] proposed a cost model for allocation of reliabilities during the design phase by minimizing a cost function, which depends on fixed development costs and on a previously experienced failure decrease cost. Tamura and Yamada [12] proposed a stochastic differential equations based SRGM model to control the software development process in terms of reliability, development effort and version-upgrade time for open source software.

This work was supported in part by the National Natural Science Foundation of China under Grants 61100127 and 61174170, the National Research Foundation for the Doctoral Program of Higher Education of China under Grant 20120111110001, and the Anhui Provincial Natural Science Foundation under Grants 1508085MF132 and 1508085QF129.

* Corresponding author at: School of Computer and Information, Hefei University of Technology, Hefei 230009, China.
E-mail addresses: Yuefeng@hfut.edu.cn (F. Yue), zg@hfut.edu.cn (G. Zhang), szp@hfut.edu.cn (Z. Su), Luyang.lu@126.com (Y. Lu), creyer@163.com (T. Zhang).

http://dx.doi.org/10.1016/j.neucom.2014.09.103
0925-2312/© 2015 Elsevier B.V. All rights reserved.

Please cite this article as: F. Yue, et al., Multi-software reliability allocation in multimedia systems with budget constraints using Dempster–Shafer theory and improved differential evolution, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2014.09.103
Meanwhile, many approaches for software reliability allocation have also been proposed in literature. Guan et al. [13] formulated an architecture-based approach for modeling software reliability optimization problem, and illustrated a dynamic programming algorithm to allocate the reliability to each component so as to minimize the cost of designing software while meeting the desired reliability goal. Pietrantuono et al. [14] proposed a reliability and testing resources allocation model called Discrete Time Markov Chain (DTMC)-type state-based model, which aimed to quantitatively identify the most critical components of software architecture in order to best assign the testing resources to them. Chatterjee et al. [15] first established a software system hierarchy, which combines the user’s view of the system with that of the software manager and the programmer, then use fuzzy analytic hierarchy process (FAHP) [16] to derive the required model parameters from the hierarchy. Ahmadian and Soltanpanah [17] dealt with a reliability optimization problem for a series system with multiple-choice and budget constraints, and developed ant colony optimization (ACO) for the problem. Sabinineni and Kurra [18] used Dynamic Algorithm for the purpose of allocation of reliability of components in a software product Design phase. Hou et al. [19] established a fuzzy multi-objective software reliability allocation model, and proposed a Multi-Objective Estimation of Distribution-Bacterial Foraging Algorithm (MEDA-BFA) based on estimation of distribution to solve the model. Kaveh et al. [20] proposed a new dynamic self-adaptive multi-objective particle swarm optimization(DSAMOPSO) method to solve binary-state multi-objective reliability redundancy allocation problems (MORAPs). Tavakkoli-Moghaddama et al. [21] proposed a genetic algorithm (GA) for a redundancy allocation problem for the series-parallel system.

From these literatures above, the area of software reliability allocation has taken the dual and often conflicting constraints of maximizing reliability and minimizing cost into account, and there are two aspects of views: one is searching an optimal reliability allocation method to achieve the given reliability such that the development cost can be as small as possible; the other is on the premise of the given cost so that the software reliability can be maximized. However, the existing work is mainly for a single software, and not suitable to deal with the case of multiple softwares. Moreover, the existing approaches mostly adopted AHP method to derive the required model parameters, and not consider the uncertainty and incompleteness in practical engineering.

Therefore, the aim of this paper is to propose a model of architecture-based multi-software reliability allocation, address before developing the software searching an optimal reliability allocation method on the premise of the given budget so that the multi-software utility can be maximized. On this condition, considering the uncertainty and incompleteness, we adopt Dempster–Shafer theory (DST) [29,22,31,24] to identify relative reliability weights, and develop an efficient differential evolution (DE) algorithm [25–27] to solve the model.

The remaining part of this paper is organized as follows: Section 2 formalizes the architecture-based multi-software Budget-Constrained Reliability-maximization (MSBCRM) model. Section 3 introduces how to identify relative reliability weights based on Dempster–Shafer Theory. In Section 4, we present the solution algorithm for the model. In Section 5, we evaluate its performance by contrast experiments. Finally, Section 6 concludes the paper.

2. The model

In this section, we discuss the hierarchy of multi-software reliability, give the definition of software utility, and formulate the architecture-based multi-software Budget-Constrained Reliability-maximization model.

2.1. Hierarchy of multi-software

In Fig. 1, the hierarchy of multi-software reliability is mainly based on the work of [7,15]. The hierarchy is a top-down approach, which starts from the top with the user’s view, which has been defined as the overall reliability goal of each software $S_i$, denoted by $R_i (i = 1, 2, \ldots, p)$.

The user of each software bases his assessment on the functionality and attributes of the software, which are represented at the second level of the hierarchy. The user expects the software to perform a set of functions and produce the desired result. This is the users view about each software. Considering the repetitive functions among similar softwares, the second level of the hierarchy denotes $f$ functions that the users of all softwares have enumerated, denotes them by $F_k (k = 1, 2, \ldots, f)$.

The third level of the hierarchy is the computer program written by software engineers to accommodate the functions specified by users, and denotes them by $P_i (i = 1, 2, \ldots, n)$. This level denotes the software engineers (SE) view of the software. Generally, each user-specified function would be programmed into more than one programmes.

The fourth level of hierarchy contains the independent modules of which the programmes are composed. In the formulation, it is assumed that SE adheres strictly to the concept of structured programming, which has become an inevitable programming approach for medium and large systems, which in turn, are prime candidates for using a reliability allocation model. As same as [7] and [15], in this paper, it is assumed that the modules are independent units which themselves may have submodules, but each submodule belongs to only one module. The independent modules are denoted by $M_j (j = 1, 2, \ldots, m)$. Hierarchical structure is stopped at the level of the independent modules.

On the whole, the hierarchy can well link the user’s view about reliability to the software manager’s and programmer’s view of the software.

![Fig. 1. Hierarchy of multi-software reliability.](image-url)
Note that the hierarchy accords with the natural state of affairs in object oriented programming (OOP) [28]. In the OOP approach, one can divide the software into independent “classes”, which have their own subclasses, and objects are instances of a class or subclass. Each independent class in OOP forms a module.

2.2. Software utility

Software utility has been defined as how reliably the user can perform various functions within the software [7]. In general, users may attach different weights to the software functions, so the software’s function reliability must be weighted by the relative importance that the user attaches to each function. It is often considered software utility is a linear function of software attributes (functions) and the attributes’ reliability as

\[
U_l = \sum_{F_i \in S_l} w_{P_i} \cdot r_{F_i}
\]

where \(U_l\) is the utility of software \(S_l\), \(r_{F_i}\) is the reliability of function \(F_i\), and \(w_{P_i}\) is the global relative weight of function \(F_i\).

As Fig. 1 has linked the user’s view about reliability to the software manager’s and programmer’s view of the software in the third level, user’s utility of software can be alternatively defined as the combination of the programme reliability and the relative importance that the software managers or programmers attach to each programme as

\[
U_l = \sum_{P_i \in S_l} w_{P_i} \cdot r_{P_i}
\]

where \(r_{P_i}\) is the reliability of program \(P_i\) and \(w_{P_i}\) is the global relative weights of programme \(P_i\).

Similarly, in the fourth level, the user’s utility of software is defined as

\[
U_l = \sum_{M_j \in S_l} w_{M_j} \cdot r_{M_j}
\]

where \(r_{M_j}\) is the reliability of module \(M_j\) and \(w_{M_j}\) is the global relative weights of module \(M_j\).

As the modules are independent units, \(r_{M_j}\) of programme \(P_i\) can be denoted as

\[
r_{P_i} = \prod_{M_j \in P_i} r_{M_j}
\]

Using Eq. (4) in Eq. (2) yields

\[
U_l = \sum_{M_j \in S_l} w_{M_j} \cdot \prod_{M_j \in P_i} r_{M_j}
\]

Eq. (5) replaces the software utility from the user’s view with the reliability from the software manager’s and programmer’s view.

2.3. The model

Based on the hierarchy of multi-software and the definition of software utility, multi-software reliability allocation is how to allocate the reliability among different modules subject to a budget constraint (including human, material resources, etc.), so that multi-software utility can be maximized. This can be described by the model

\[
\text{Max } U = \sum_{i=1}^{n} \left[ w_{P_i} \cdot \prod_{M_j \in P_i} r_{M_j} \right]
\]

In MSBCRM model, the objective function is to maximize the utility \(U_l\) of each software \(S_l\) and \(r_{M_j}\) are the only unknown variables. Once \(r_{M_j}\) are determined, we can compute the programme reliability allocations from (4). Hence, the model yields reliability allocations for the third and fourth level of the hierarchy.

The constraint (7a) refers to the reliability of \(M_j\), where \(u_i\) is the “feasible” level of reliability, and \(l_i\) is the “minimum acceptable” level of reliability. The “feasible” and “minimum acceptable” levels of reliability are the technical limits which should be determined at the planning or design stage of the software development, and \(u_i, l_i \in [0, 1]\).

The constraint (7b) shows the relation between the cost of attaining a reliability level in a module and its budget on the assumption that reliability and budget have a linear relationship. For module \(M_j\), \(a_{M_j}\) is the fixed overhead cost for attaining reliability \(r_{M_j}\), and \(b_{M_j}\) is the variable cost, which is the marginal cost of increasing the reliability of \(a_{M_j}\) by one unit. On the right-hand side of the inequality, \(B_{Sl}\) is the budget of software \(S_l\), \(w_{M_j} \cdot \sum_{j=1}^{n} B_{Sl}\) is the maximum budget of module \(M_j\), and its cost must not exceed this level.

The constraint (7c) refers to the relation between the cost of attaining a reliability level in a software and its budget. On the left-hand side of the inequality, \(\sum_{M_j \in S_l} (a_{M_j} + b_{M_j} \cdot r_{M_j})\) is the total cost of all modules included in software \(S_l\), which must remain below the available budget \(B_{Sl}\).

Note that in this model, \(w_{P_i}\) and \(w_{M_j}\) are the global relative weights of programme \(P_i\) and module \(M_j\), respectively. However, as the users’ view of the software is external and ends at the functional level, they are hardly able to directly give his preference for each programme and module. Likewise, the programmer sees only the third and fourth levels. Therefore, to solve the model, our primary task is how to identify the relative importance of each programme \(P_i\) at the third level and each module \(M_j\) at the fourth level of the hierarchy combining the users’ assessment of the softwares at the first and second levels with the programmers’ view at the third and fourth levels. In this paper, we will first adopt Dempster–Shafer theory to achieve the task, and then use differential evolution algorithm to obtain the optimal reliabilities of the components to achieve the highest utility for each software.

3. Identifying relative reliability weights based on Dempster–Shafer Theory

The existing works mostly adopted the analytic hierarchy process (AHP) to identify relative reliability weights. Its main step is to establish the judgement matrix by pairwise comparisons among decision elements in each level. However, the complexity of the judgement matrix establishment will exponentially increase along with the increase of decision elements number. Due to the complexity and uncertainty involved in real-world decision problems, one may sometimes prefer fuzzy judgements to crisp comparisons, and judgements may be incomplete. In this condition, AHP is not an effective means. Therefore, in this paper, we adopt Dempster–Shafer theory (DST) to identify relative reliability weights.

3.1. Dempster–Shafer theory

The Dempster–Shafer theory (DST) was first developed by Dempster [29,30], and later extended and formalized by Shafer [31,23]. It
allows one to combine evidence from different sources and arrive at a degree of belief which takes into account all the available evidence, and has been widely applied in artificial intelligence, expert systems, pattern recognition, information fusion, risk assessment, multiple attribute decision analysis, etc. [22,32–37]. In DST, the basic unit of knowledge representation is called a basic probability assignment (Bpa) function.

**Definition 1.** Assume that the symbol \( U_X \) is the set of possible values of a variable \( X \), \( U_X \) is called the frame for \( X \).

**Definition 2.** A Bpa function \( \mu \) for \( X \) can be represented with the following three equations:

\[
\mu : 2^{U_X} \rightarrow [0, 1]
\]

\[
\mu(\emptyset) = 0
\]

\[
\sum_{A \in 2^{U_X}} \mu(A) = 1
\]

where \( 2^{U_X} \) denotes the set of all nonempty subsets of \( U_X \).

The value of \( \mu(A) \) expresses the proportion of all relevant and available evidence that supports the claim that a particular element of \( X \) (the universal set) belongs to the set \( A \) but to no particular subset of \( A \).

When the data is coming from different multiple sources that provide different assessments for the same frame of discernment, combination rules must be used to aggregate information. Therein the Dempster rule is critical to the original conception of DST.

**Definition 3.** Assume that \( \mu_1(B) \) and \( \mu_2(C) \) are two different and independent evidences, and Dempster rule is purely a conjunctive operation (AND), and combination \( \mu_{12} \) is calculated in the following manner:

\[
\mu_{12}(A) = \left\{ \begin{array}{ll}
\sum_{B \cap C = A} \mu_1(B) \cdot \mu_2(C) / (1 - K) & \text{when } A \neq \emptyset \\
0 & \text{when } A = \emptyset
\end{array} \right.
\]

where

\[
K = \sum_{B \cap C = \emptyset} \mu_1(B) \cdot \mu_2(C)
\]

From (12), \( K \) is calculated by summing the products of the Bpas of all sets where the intersection is null, and represents basic probability mass associated with conflict. The normalization factor \( 1 - K \) has the effect of completely ignoring conflict and attributing any probability mass associated with conflict to the null set. And Dempster rule has been proved to be commutative, associative, but not idempotent or continuous.

### 3.2. Expression of uncertainty and incompleteness

In practical engineering, for users, it is difficult to give the importance of each software function by a precise value, as well as programmers. Contrarily to the crisp expression, one tends to fuzzy judgement, such as “more important”, “important” and “less important”. And these judgement can well accord with the expression of human beings.

Here, we adopt “rank” to rationally express the uncertainty of users’ or programmers’ view in a fuzzy manner.

**Definition 4.** The rank \( H \) is defined as levels of importance as follows:

\[
H = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}
\]

where 0 means no importance, while 1 means the most importance. According to **Definition 4**, one’s judgement can be expressed as \( (f^1, \lambda_f^1) \), where \( f^1 \) may be a function, programme, and module, and \( \lambda_f^1 \) is its importance from the user’s or programmer’s view. The higher the value of \( \lambda_f^1 \), the more important the \( f^1 \).

For example, assume \( S_1 \) has two function \( F_1 \) and \( F_2 \), and \( S_2 \) has two function \( F_2 \) and \( F_3 \), the preferences of user1 and user2 can be respectively expressed as

\[
S_1 : \{(F_1, 0.3), (F_2, 0.7), (F_3, 0.1)\}
\]

\[
S_2 : \{(F_1, 0), (F_2, 0.6), (F_3, 0.3)\}
\]

Note that, users or programmers do not rigidly adhere to these values in \( H \). The main role of \( H \) is to guide the judging more consistent. And one’s preferences may not add to 1. When the sum is 1, it corresponds to a traditional probabilistic representation. Then when the sum is less than 1, it called the “incomplete case”, this implies the user’s uncertainty among different functions, which cannot be dealt with traditional AHP methods. Hence, in such application, to achieve the goal, DST is a suitable and applicable way to combine the different views for identifying the relative reliability weights of function, programme and modules of each level with the users assessment about the multi-software.

#### 3.3. Algorithm for identifying relative reliability weights based on DST

Now we take the second level, for an example, to give the algorithm (see **Algorithm 1**) for identifying relative reliability weights for each element based on DST.

**Algorithm 1.** The algorithm for identifying relative reliability weights.

1: for \( k=1 \) to \( p \) do
2: for \( k=1 \) to \( p \) do
3: compute \( \mu_2(F_k) = W_{S_1} \cdot \lambda_{F_k} \)
4: end for
5: end for
6: \( \mu_1(F) = 1 - \sum_{k=1}^{p} \mu_2(F_k) \)
7: \( \mu_{12}(F) = \sum_{k=1}^{p} \mu_1(F_k) \)
8: for \( k=1 \) to \( p \) do
9: compute \( \mu_2(F_k) \)
10: set \( \mu_{12}(F_k) = \mu_1(F_k) \)
11: end for
12: end for
13: end for

Here, \( W_{S_1} = (B_{S_1}) / \sum_{k=1}^{p} B_{S_1} \). Similarly, we can determine the relative importance of each programme \( P_1 \) at the third level and each module \( M_1 \) at the fourth level. It is worth mentioning that the relative weights of the elements in various levels must be aggregated with respect to that at the top level of the hierarchy.

### 4. The solution

Generally, the number of the solution for the MSBCRM model is more than one, and traditional searching methods (such as branch-and-bound method, cutting plane method) is difficult to achieve
optimal solution for large-size problems. Moreover, in the MSBCM model, our goal is just to determine each module's reliability to maximize multi-software utility subject to a budget constraint, which is similar to a combinatorial optimization problem. Therefore, in this work, we present an efficient differential evolution algorithm to resolve the proposed model.

4.1. The differential evolution algorithm

Differential evolution (DE) algorithm [25–27], proposed by Storn and Price, is an efficient and effective global optimizer in the continuous search domain. In this paper, DE/best/1/exp with three main operations (mutation, crossover and selection) is adopted, whose flowchart is shown in Fig. 2.

In DE, population is the set of candidate solutions, referred to individual, for an optimization problem. DE utilizes the competition and cooperation of individuals in population to guide search process.

4.2. The basic solution algorithm based on DE

For the MSBCM model, we use one-dimensional real encoding shown as Fig. 3 to express an individual. For each encoding, there are $m$ real variable, and $r_M$ is the reliability of module $M_j$.

The fitness function of individual $l$ is defined as

$$f(l) = U$$  \hspace{1cm} (15)

From (6), the bigger the fitness $f(l)$, the better the solution $l$.

Assume $N$ is the size of each population, $l_G$ denotes the $G$th population, $l^C_G = \{l^C_1, \ldots, l^C_M\}$ is the $G$th individual of population $l_G$, $l^i_G = \{r^i_M, \ldots, r^i_M\}$ is the mutation individual, $l^1_G = \{r^1_M, \ldots, r^1_M\}$ and $l^2_G = \{r^2_M, \ldots, r^2_M\}$ are the two crossover individuals. The solution algorithm based on DE is described in Algorithm 2, where $F$, called the scale factor, is a real constant in the interval $[0, 2]$, and $CR \in [0, 1]$ is the crossover factor which controls the number of components taken from the mutant vector.

**Algorithm 2.** The solution algorithm based on DE.

1: Set iteration number $G = 0$
2: // Generate $N$ random solutions using uniform distribution;
3: for $i = 1$ to $N$
4: for $j = 1$ to $m$
5: $r^0_{Mj} = \text{rand}j(0, 1)$
6: end for
7: Calculate $f(l^0_G)$
8: end for
9: for $G = 0$ to $G_{\text{max}}$
10: // Implement the mutation operation;
11: for $i = 1$ to $N$
12: randomly select different individual $l^1_G$, $l^2_G$, $l^3_G$ from $l_G$
13: set $l^C_{\text{best}} = \max f(l^C_1), f(l^C_2), f(l^C_3)$
14: set $l^i_G$ according to $l^i_G = l^C_{\text{best}} + F \cdot (l^C_{g2} - l^C_{g3})$
15: end for
16: // Implement the crossover operation;
17: for $i = 1$ to $N$
18: for $j = 1$ to $m$
19: set $\text{rand} = \text{rand}d(0, 1)$
20: if $\text{rand} \leq CR$
21: set $r^1_{Mj} = r^i_{Mj}$
22: else
23: set $r^1_{Mj} = r^C_{Mj}$
24: end if
25: set $r^2_{Mj} = l^i_{Mj}$
26: end for
27: end for
28: // Implement the selection operation;
29: for $i = 1$ to $N$
30: Calculate $f(l^i_{G1})$ and $f(l^i_{G2})$
31: set $f(*) = \min f(l^i_{G1}), f(l^i_{G2})$
32: set $l^i_{G+1} = *$
33: end for
34: end for
35: end for

It is worth mentioning that individuals generated by initialization, mutation and crossover operations may have to be faced with three tough problems:

1. If for $\exists j \in \{1, \ldots, m\}$, $r^C_{Mj} < l_j$ or $r^C_{Mj} > u_j$, that is, the reliability of $M_j$ is not to satisfy the constraint (7a), here the encoding is invalid.

2. If for $\exists j \in \{1, \ldots, m\}$, $a_{Mj} + b_{Mj} \cdot r_{Mj} > w_{Mj} \cdot \sum_{i=j}^P B_{S_i}$, that is, the cost of module $M_j$ exceeds its maximum budget, here, the encoding is also invalid.

3. If for $\exists j \in \{1, \ldots, m\}$, $\sum_{M \in S_j} (aM + bM \cdot r_{M}) > BS_j$, that is, the total cost of software $S_j$ remains above its available budget $BS_j$, here the encoding is invalid.

**Fig. 2.** Flowchart of DE.

**Fig. 3.** one dimensional real encoding.
In fact, as long as either of problems shown above takes place, this encoding is just invalid, which consumedly debases the availability of encodings, and results in much low evolution efficiency. Therefore, in the next section, we will present a repairing algorithm to improve evolution efficiency, and give the solution algorithm based on DE with encoding repairing (hence called DEER).

4.3. The solution algorithm based on DE with encoding repairing

Assume \( r = [r_1, \ldots, r_m, \ldots, r_{m_n}] \) denotes an invalid encodings, the repairing algorithm can be described as Algorithm 3.

Algorithm 3. The algorithm for encoding repairing.

1: for \( j = 1 \) to \( m \) do
2: if \( r_m < l_j \) or \( r_m > u_j \) then
3: set \( r_m = l_j + \text{rand}(0, 1) \cdot (u_j - l_j) \)
4: end if
5: while \( a_m + b_m \cdot r_m > w_m \cdot \sum_{i=1}^{p} B_i \) and \( r_m \geq l_j + \varepsilon \) do
6: set \( r_m = r_m - \varepsilon \)
7: end while
8: end for
9: for \( l = 1 \) to \( p \) do
10: for \( \forall M_j \in S_{ij} \) do
11: Flag\( (M_j) = 0 \)
12: end for
13: while \( \sum_{M_j \in S} (a_{M_j} + b_{M_j} \cdot r_{M_j}) > B_j \) do
14: select \( M_j \) with the lowest \( w_{M_j} \) and Flag\( (M_j) = 0 \)
15: if \( r_{M_j} \geq l_j + \varepsilon \) then
16: set \( r_{M_j} = r_{M_j} - \varepsilon \)
17: else
18: set Flag\( (M_j) = 1 \)
19: end if
20: end while
21: end for

where \( \varepsilon \) is the adjustment step.

The DEER algorithm is shown in Algorithm 4, which combines Algorithm 2 and 3.

Algorithm 4. The solution algorithm based on DE with encoding repairing.

1: Set iteration number \( G = 0 \)
2: Generate \( N \) random solutions using uniform distribution;
3: Implement encoding repairing for \( N \) individuals
4: for \( G = 0 \) to \( G_{\text{max}} \) do
5: Implement the mutation operation;
6: Implement the crossover operation;
7: Implement encoding repairing for \( 2N \) crossover individuals
8: Implement the selection operation;
9: end for

5. Performance evaluation

In this section, we will evaluate the performances of the algorithm for identifying relative reliability weights based on DST, the MSBCRM model and the proposed solution algorithm, respectively.

5.1. Experimental setting

In the experiment, we just consider an example with three multi-media software systems: an audio retrieval system \( S_1 \), a video retrieval system \( S_2 \), and an image retrieval system \( S_3 \), the interface is shown as Fig. 4. Assume that the budget of each software is shown in Table 1. Users of the three system have enumerated their required functions:

Image retrieval system which functions (1) to execute an image query. (2) To show the “most” similar images between its visual feature and the images’ feature.

Audio retrieval system which functions (1) to execute an audio query. (2) To show the “most” similar audio between its feature and the audios’ feature.

Video retrieval system which functions (1) to execute an audio query. (2) To execute an image query. (3) To show the “most” similar image frames between its image feature and the video frames' feature.

By analyzing users' specified functions of the three systems, software engineers have decided to written several computer program to accommodate each function. Take the video retrieval system for example, software engineers consider that three programmes can achieve the two functions: visual search engine, web server, and user interface. Furthermore, they also give the independent composed modules for each programme as follows.

Visual search engine, which consists of (1) to extract visual features from query image and image dataset. (2) Given a query image, to find out the “most” similar images between its visual feature and the images' feature in a dataset. (3) To communicate with the web server, that is to get the query image and to return the results list in a way that could be more readable by users.

Web server, which consists of (1) to get the query image from user query. (2) To communicate with visual search engine. (3) To pass the query image and obtain the query result.

User interfaces, which consists of (1) the translation of a human query into a set of parameters required by the search engine. (2) To show the retrieved content for browsing.

Generally, the hierarchical structure of the 3 softwares is shown in Fig. 5, where the number of each line indicates that it is a local relative weight of the lower element corresponding to the upper one, which is given by its users or software engineers subjectively.

For the standard DE and the DE with encoding repairing, set \( N = 40, G_{\text{max}} = 500, F = 0.5 \), and \( cr = 0.94 \). Simulations are run with VC++ on a PC with an Intel Core Duo 2 GHz CPU, and then the statistical results are calculated and compared.

5.2. Evaluation of the algorithm for identifying relative reliability weights based on DST

Note that, in Fig. 5, users or programmers' preferences may not add to 1. For example, programmers have decided to use program \( P_1 \) and \( P_2 \) to accomplish the function \( F_1 \), and gives view about the importance 0.5 for \( P_1 \) and 0.35 for \( P_2 \). Here, it is an incomplete case. That is, programmers have uncertainty about \( P_1 \) and \( P_2 \), which is very difficult to deal with by AHP method. However, DST can do. Now we take the function level, for an example, to illuminate the process of identifying relative reliability weights.

- The views of each user can be denoted by

  \[ S_1 : \{(F_1, 0.4), (F_2, 0.5), (F_3, 0), (F_4, 0)\} \]
  \[ S_2 : \{(F_1, 0), (F_2, 0.4), (F_3, 0.2), (F_4, 0.1)\} \]
  \[ S_3 : \{(F_1, 0), (F_2, 0), (F_3, 0.4), (F_4, 0.5)\} \]

- Calculate the \( \mu_1(F_1) \) and \( \mu_1(F_2) \) as Table 2.
First combine the judgement of User 1 and User 2, and then combine with that of User 3. The results are shown in Table 3.

Obtain the globe relative weights for each function by normalization:

\[
WF = \left[ \frac{1}{2} 0.0977, \frac{1}{2} 0.3696, \frac{1}{2} 0.2486, \frac{1}{2} 0.2841 \right]
\]

Similarly, we can calculate the global relative weights of program and module:

\[
WF = \left[ 0.2614, 0.0283, 0.5103, 0.0590, 0.1410 \right]
\]

\[
WF = \left[ 0.2155, 0.0340, 0.3937, 0.2097, 0.0737, 0.0428, 0.0306 \right]
\]

As in practical situation, one inclines to give his preference in a qualitative methods, which may be with a certain degree of uncertainty, subjectivity, and incompleteness. Conventional aggregation methods is difficult to deal with such problem. From the process above, the aggregation is in compliance with human thinking, and is flexible and effective.

5.3. Evaluation of MSBCRM model

Zahedi and Ashrafi [7] developed a classical model of single software reliability allocation (Hence called SSRA). The subsequent main literature for solving software reliability allocation are almost completely based on this model. In contrast, the model MSBCRM in this paper is presented to analyze the reliability allocation problem of multiple softwares in multimedia systems. Therefore, to illustrate the performance of the multi-software model against the single software model, we choose the representative SSRA for comparison.

First, as the global relative weights of programmes and modules have been obtained by using DST method which are shown as Table 1.

Figure 5. The hierarchy of 3 softwares

Please cite this article as: F. Yue, et al., Multi-software reliability allocation in multimedia systems with budget constraints using Dempster–Shafer theory and improved differential evolution, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2014.09.103
and (18), respectively, the MSBCRM model is formulated as

\[
\text{Max } U = 0.2614(r_M \cdot r_M) + 0.0283(r_M \cdot r_M)
\]

\[
\cdot (r_M) + 0.5103(r_M \cdot r_M) + 0.0590(r_M \cdot r_M)
\]

\[
\cdot (r_M) + 0.141(r_M \cdot r_M \cdot r_M)
\]

(20)

Subject to

\[
0.6 \leq r \leq 1, j = 1, 2, \ldots , 7
\]

\[
15 + 25 \cdot r_M \leq 0.215 \cdot 1000
\]

\[
14 + 28 \cdot r_M \leq 0.0340 \cdot 1000
\]

\[
59 + 55 \cdot r_M \leq 0.3937 \cdot 1000
\]

\[
41 + 40 \cdot r_M \leq 0.2097 \cdot 1000
\]

\[
7 + 35 \cdot r_M \leq 0.0737 \cdot 1000
\]

\[
16 + 29 \cdot r_M \leq 0.0428 \cdot 1000
\]

\[
10 + 12 \cdot r_M \leq 0.0306 \cdot 1000
\]

\[
(15 + 25 \cdot r_M) + (14 + 28 \cdot r_M) + (59 + 55 \cdot r_M) + (41 + 40 \cdot r_M) \leq 25
\]

\[
(15 + 25 \cdot r_M) + (14 + 28 \cdot r_M) + (59 + 55 \cdot r_M) + (41 + 40 \cdot r_M)
\]

\[
+ (7 + 35 \cdot r_M) + (16 + 29 \cdot r_M) + (10 + 12 \cdot r_M) \leq 45
\]

\[
(59 + 55 \cdot r_M) + (41 + 40 \cdot r_M) + (16 + 29 \cdot r_M) + (10 + 12 \cdot r_M) \leq 30
\]

Next, we solve the software reliability allocation problem in a serial manner based on the SSRA model. Assume that the order is: first \( S_1 \), second, \( S_2 \), and the last \( S_3 \). Since a module may be shared by multiple softwares, a module, which has been allocated a reliability in the former software, is viewed as a known quantity in the latter software. Assume that the global relative weights of programmes and modules in each software have been obtained by using DST method, the SSRA models for 3 softwares is formulated as

\[
U = 0.25 U_1 + 0.45 U_2 + 0.3 U_3
\]

(22)

\[
\text{Max } U_1 = 0.6086(r_M \cdot r_M) + 0.1442(r_M \cdot r_M)
\]

\[
\cdot (r_M) + 0.2472(r_M \cdot r_M)
\]

(23)

Subject to

\[
0.6 \leq r \leq 1, j = 1, 2, \ldots , 7
\]

\[
15 + 25 \cdot r_M \leq 0.6618 \cdot 250
\]

\[
14 + 28 \cdot r_M \leq 0.1174 \cdot 250
\]

\[
59 + 55 \cdot r_M \leq 0.1417 \cdot 250
\]

\[
41 + 40 \cdot r_M \leq 0.0791 \cdot 250
\]

(24)

\[
\text{Max } U_2 = 0.2966(r_M \cdot r_M) + 0.5391(r_M \cdot r_M) + 0.0463(r_M
\]

\[
\cdot r_M \cdot r_M) + 0.1180(r_M \cdot r_M \cdot r_M)
\]

(25)

Subject to

\[
0.6 \leq r \leq 1, j = 1, 2, \ldots , 7
\]

\[
59 + 55 \cdot r_M \leq 0.3970 \cdot 450
\]

\[
41 + 40 \cdot r_M \leq 0.2215 \cdot 450
\]

\[
7 + 35 \cdot r_M \leq 0.0490 \cdot 450
\]

\[
16 + 29 \cdot r_M \leq 0.0283 \cdot 450
\]

\[
10 + 12 \cdot r_M \leq 0.0205 \cdot 450
\]

Finally, we use the proposed algorithm DEER to solve the MSBCRM and SSRA model. Table 4 gives the contrast results. This table shows that the MSBCRM model can obtain a better solution than the SSRA model as a whole, though the utility of \( S_1 \) obtained by the MSBCRM model is lower than that by the SSRA. The reason is that the SSRA model is in a serial manner, and give priority to the first solved softwares, which may sacrifice the utility of the subsequent softwares, and result in the circumstances where the more backward, the lower the utility of the software. Meanwhile, the MSBCRM model is in a parallel manner, and consider all of softwares simultaneously, and aims at the maximum of the total utility.

Table 4

<table>
<thead>
<tr>
<th>Element</th>
<th>MSBCRM</th>
<th>SSRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_M )</td>
<td>0.6</td>
<td>1</td>
</tr>
<tr>
<td>( r_M )</td>
<td>0.6</td>
<td>1</td>
</tr>
<tr>
<td>( r_M )</td>
<td>0.896</td>
<td>0.616</td>
</tr>
<tr>
<td>( r_M )</td>
<td>0.998</td>
<td>0.853</td>
</tr>
<tr>
<td>( r_M )</td>
<td>1</td>
<td>0.955</td>
</tr>
<tr>
<td>( r_M )</td>
<td>0.924</td>
<td>0.647</td>
</tr>
<tr>
<td>( r_M )</td>
<td>1</td>
<td>0.989</td>
</tr>
<tr>
<td>( U_1 )</td>
<td>0.5175</td>
<td>0.81426</td>
</tr>
<tr>
<td>( U_2 )</td>
<td>0.7362</td>
<td>0.6698</td>
</tr>
<tr>
<td>( U_3 )</td>
<td>0.8946</td>
<td>0.5322</td>
</tr>
<tr>
<td>( U )</td>
<td>0.744732</td>
<td>0.6645</td>
</tr>
</tbody>
</table>

5.4. Results and comparisons between the standard DE and the DEER

In this section, to evaluate the proposed encoding repairing algorithm, we compare the DEER algorithm with the standard DE by solving the MSBCRM model. The contrast results are shown in Table 5. The contrast results show that DEER can obtain better solution within less run time than DE, and make the module with higher weight have more reliability. The reason is that DEER repairs every invalid encoding into a valid one, and results in no discarded encodings, while DE discards many encodings which may contain valuable information in the evolving process, and results in much low evolution efficiency.

5.5. Results and comparisons between DEER and typical algorithms

In this experiment, we compare our DEER algorithm with dynamic programming (DP) in [18], particle swarm optimization (PSO) in [20], and genetic algorithm (GA) in [21]. Fig. 6 gives the evolving curve of the best solution for different algorithms. Table 6 presents the contrast results of utility objective values and run time in different algorithms.

From Fig. 6 and Table 6, the utility objective value of the optimal solution searched by DEER is 0.744732, which is better than DP, PSO and GA. Since DEER employs mutation, crossover, selection operators, it consumes a litter more time than PSO, but less than GA and DP. On the whole, DEER has been able to get a very good solution in a reasonable CPU time.

6. Conclusion

In this paper, we have addressed the multi-software reliability allocation issue in a multimedia platform with multi-softwares and

Table 5

<table>
<thead>
<tr>
<th>Element</th>
<th>DEER</th>
<th>DE</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_M )</td>
<td>0.6</td>
<td>0.644</td>
</tr>
<tr>
<td>( r_M )</td>
<td>0.6</td>
<td>0.606</td>
</tr>
<tr>
<td>( r_M )</td>
<td>0.896</td>
<td>0.915</td>
</tr>
<tr>
<td>( r_M )</td>
<td>0.998</td>
<td>0.94</td>
</tr>
<tr>
<td>( r_M )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( r_M )</td>
<td>0.924</td>
<td>0.923</td>
</tr>
<tr>
<td>( r_M )</td>
<td>1</td>
<td>0.974</td>
</tr>
<tr>
<td>( U_1 )</td>
<td>0.5175</td>
<td>0.5253</td>
</tr>
<tr>
<td>( U_2 )</td>
<td>0.7362</td>
<td>0.7246</td>
</tr>
<tr>
<td>( U_3 )</td>
<td>0.8946</td>
<td>0.8695</td>
</tr>
<tr>
<td>( U )</td>
<td>0.744732</td>
<td>0.732262</td>
</tr>
<tr>
<td>Run time (s)</td>
<td>0.031</td>
<td>0.343</td>
</tr>
</tbody>
</table>

Please cite this article as: F. Yue et al., Multi-software reliability allocation in multimedia systems with budget constraints using Dempster–Shafer theory and improved differential evolution, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2014.09.103
solve the problem with budget constraints using Dempster–Shafer theory and differential evolution with encoding repairing. We have formulated the problem as a nonlinear programming model subject to a number of given constraints. To solve the model, we have shown how to use Dempster–Shafer theory to establish the global relative weights of each element, and also developed an efficient differential evolution algorithm with encoding repairing for finding best solutions. To evaluate the performance of the proposed algorithm, its performance has been benchmarked against three available algorithms in the literature. Comparisons results show that the proposed approach is effective and efficient for solving the reliability optimization problem.

For future work, we will concentrate on considering other project-specific production constraints, such as constraints on staffing resources and time constraints for project completion, in the multi-software reliability optimization problem.

References

Guofu Zhang received the B.S. and Ph.D. degrees in computer science from Hefei University of Technology, Hefei, China, in 2002 and 2008, respectively. He is currently an Associate Professor with School of Computer and Information, Hefei University of Technology. He has published more than 20 papers in high impact factor journals and conference proceedings. His research interests include evolutionary computation, intelligent agents, and agent-based systems integration and collaboration.

Zhaopin Su received the B.S. and Ph.D. degrees in computer science from Hefei University of Technology, Hefei, China, in 2004 and 2008, respectively. She is currently an Associate Professor with School of Computer and Information, Hefei University of Technology. She is interested in agent based simulation, coalition formation for disaster response, and collaborative decision support systems for disaster response management.

Yang Lu received the B.S. degree in electronic engineering from Shanghai Jiao Tong University, Shanghai, China, in 1988, and the Ph.D. degree in computer science from Hefei University of Technology, Hefei, China, in 2002. He is currently a Professor with School of Computer and Information, Hefei University of Technology. His research interests include software reliability engineering, wireless sensor networks, and intelligent visual surveillance. He has published over 60 research papers.

Ting Zhang received the B.S. degree from Foshan University, Foshan, China, in 2012. She is currently working toward the M.S. degree with School of Computer and Information, Hefei University of Technology. Her research interests include evolutionary computation and software engineering.