Selection of the optimal set of versions of N-version software using the ant colony optimization

A Aljarbouh
University of Central Asia, Lenin Street, 310, Naryn, 722918, Kyrgyz Republic
E-mail: ayman.aljarbouh@ucentralasia.org

Abstract. The article discusses the problem of ensuring a high level of software reliability. A software reliability high level can be ensured using N-version programming. N-version software is distinguished by the software components redundancy, designed to solve one problem using different methods. Software redundancy requires additional resources. The challenge is thus to increase the software reliability, while at the same time minimising the resources used. In this article, ant colony optimization is used to solve the problem of choosing the optimal set of versions of N-version software.

1. Introduction
Currently, there are several approaches to ensuring software reliability such as fault avoidance, fault detection, fault correction, as well as ensuring fault tolerance [1-5].

N-version programming is a software development method in which several software modules designed to solve the same problem are developed independently of each other [6, 7]. Such software modules are called versions of N-version software.

N-version software A. Avizhienis defined as functionally equivalent software components a number' an independent generation under a single software specification [8, 9]. Preferred in the N-version software development process is the software components development by programmers in separate and independent teams. At the same time, programmers should use different programming languages. This approach allows compensating and software modules in individual components to mask failures and provides fault tolerance [10]. Thus, the performance of the target functions' N-version software will be guaranteed even in the individual versions' single failures event [11, 12].

The N-version programming efficiency is determined primarily by the software components diversity degree, which minimize the same response appearance when software errors occur and software malfunctions.

Diversification into N-version software can be achieved in the following ways [13]:

- the versions created by different programmers or programmers teams;
- various methods and algorithms implementation in different versions;
- using programs logical organization different methods;
- using languages different versions or compilers different versions from the same language.
With N-version programming, N versions get identical source data. The results produced by the N versions are compared using the selection algorithm [14-16]. Typically, versions are executed in parallel [17]. However, it is also possible to consistently execute versions of N-version software.

The creation of N-version software is based on software components redundancy. This leads to an optimization problem where, along with increasing the software reliability, it is required to minimize the resources' amount used or to restrict their use within certain limits [18, 19].

To select the N-version software version most optimal set, it is proposed to use ant colony optimization (ACO), which is a polynomial algorithm for finding approximate solutions to the finding routes problem on graphs [20-22].

Ant colony optimization was first proposed by M. Dorigo in his doctoral dissertation, which was published in 1992 [23]. Ant colony optimizations provide solutions to many combinatorial problems. Ant colony optimizations an important property is non-convergence: even after many iterations, many solutions are simultaneously investigated. As a result, there are no long delays at local extrema, which allows recommending this algorithm to use for solving complex combinatorial optimization problems.

2. Methods

An ant colony can be treated as a system in which each ant functions autonomously following certain simple rules [24]. Ants can find a short path, abandoning the old, longer, path. When choosing the movement direction, each ant in the colony is guided by the distance to the next node and a special substance amount - a pheromone left on the proposed path by its predecessors.

Passing along some path, the ant also leaves a pheromone on it, increasing its amount along the way [25]. In this case, as a rule, the pheromone evaporation is also modelled over time, that is, from iteration to iteration.

The ant colony optimization following steps can be defined:

1. Initial data input and algorithm settings and variables initialization.
2. An ant transition probabilities determination from one graph vertex to another.
3. Each ant movement route generation, taking into account the transition probabilities.
4. Each traversed path resulting length determination
5. Change in the pheromone concentration on the graph edges.
6. Repeating steps 2-5 for iterations’ a specified number, or until one or more selected conditions for the algorithm end are completed.

The ant colony optimization is based on a decision rule, according to which an ant moving probability from the current vertex to available vertices each adjacent to the current one is determined. In our case, this is the probability that the $k$th ant will select the version $j$ of module $i$ at the $t$th iteration, denoted as $p_{ijk}(t)$.

Suppose that you want to get an optimal result, taking into account both reliability and cost. Then the rule looks like this:

$$p_{ijk}(t) = \frac{\tau_j(t)^\alpha \left( \frac{1}{C_j} \right)^\beta R_j^\gamma}{\sum_{j=1}^{m_i} \tau_j(t)^\alpha \left( \frac{1}{C_j} \right)^\beta R_j^\gamma},$$  \hspace{1cm} (1)

where $\tau_j$ is the pheromone amount; $R_j$ - reliability of version $j$ of module $i$; $C_j$ - cost of version $j$ of module $i$; $\alpha$ - parameters reflecting the pheromone importance in the direction choice, $\beta$ and $\gamma$ - parameters that allow determining the reliability or cost importance; $m_i$ - number of versions of module $i$.

In our case, equation (1) determines the probability of choosing one or another version of the N-version software. As in the case of any other implementation of ACO, the highest probability does not
oblige the ant to move along this path. The values obtained according to equation (1) determine with what probability the ant will follow one or another path.

The change in the phenomenon quantity in this case, taking into account the cost and reliability optimization, occurs according to the formula:

$$\tau_j(t+1) = (1 - \rho)\tau_j(t) + K_j \left( \frac{R_j}{\max(R_j, l = 1, m_i)} \right) \left( \min(C_{ij}, l = 1, m_i) \right)$$

where \(\rho\) is the pheromone evaporation rate, \(K_j\) is the ants’ number that has chosen the module \(i\) version \(j\), \(m_i\) - number of versions of module \(i\).

3. Results and discussion

The N-version software versions’ optimal set choice using the ant colony optimization is given on the single-function N-version software, a model example without redundancy. N-version software includes several modules (figure 1). Several versions are proposed for each module implementation, which is available as components off-the-shelf with known reliability and cost values. Nevertheless, let us assume that modules saving multiple versions are undesirable due to strict cost constraints and the area uncritical nature in which this software will execute:

$$\sum_{j=1}^{m_i} X_{ij} = 1, \ j = i, \ldots, n.$$  

where \(n\) – number of modules, \(m_i\) – number of versions of module \(i\), \(X_{ij}\) – Boolean variable that equals to 1 if version \(j\) is used in module \(i\), or 0 - otherwise.

![Figure 1. Structure of N-version software.](image)

Figure 1 shows that N-version software consists of nine modules. Versions available for modules of N-version software and their characteristics are presented in table 1.

**Table 1.** Initial data on cost and reliability of N-version software versions.

| Module no. | Version no. | Cost  | Reliability |
|------------|-------------|-------|-------------|
| 1          | 1           | 200   | 0.99958     |
| 1          | 2           | 198   | 0.99926     |
| 1          | 3           | 215   | 0.99962     |
| 1          | 4           | 210   | 0.99961     |
| 2          | 1           | 180   | 0.99988     |
Figure 2. The pheromone amount changing when choosing version of module 1 of N-version software.
Figure 2 shows the change in pheromone amount when choosing a version of the first module of N-version software using ACO. The largest pheromone amount corresponds to the version selected for implementation as a part of N-version software.

Table 2 presents the modules' versions that are included in the N-version software.

| Module no. | Version no. | Cost  | Reliability |
|------------|-------------|-------|-------------|
| 1          | 4           | 210   | 0.99961     |
| 2          | 3           | 125   | 0.99989     |
| 3          | 1           | 170   | 0.99990     |
| 4          | 2           | 169   | 0.99985     |
| 5          | 2           | 130   | 0.99982     |
| 6          | 1           | 300   | 0.99997     |
| 7          | 5           | 250   | 0.99992     |
| 8          | 1           | 282   | 0.99983     |
| 9          | 2           | 125   | 0.99985     |

The set of N-version software versions shown in table 1 represents the optimal set of versions, taking into account their cost and reliability.

4. Conclusion
N-version programming is one effective approach to building reliable and fault-tolerant software. N-version software performance remains at a level sufficient for the software to perform its target functions, even in the individual components' failure events that make up the N-version software. The use of N-version software is especially important for areas with increased reliability requirements.

Since the N-version software is based on software redundancy, additional resources are required to create it. In this regard, it becomes important to solve the resources optimizing the use problem while increasing the software reliability.

This article proposes a solution to the N-version software versions choosing the optimal set problem using the ant colony optimization. Based on the given structure N-version software and versions of N-version software modules available as components-off-the-shelf with known reliability and cost values, the N-version software optimal composition was chosen.

References
[1] Raghuvanshi K K, Agarwal A, Jain K and Singh V B 2021 A time-variant fault detection software reliability model SN Applied Sciences 3(1) 18 DOI: 10.1007/s42452-020-04015-z
[2] Xiao H, Cao M and Peng R 2020 Artificial neural network based software fault detection and correction prediction models considering testing effort Applied Soft Computing Journal 94 106491 DOI: 10.1016/j.asoc.2020.106491
[3] Yi Sh and Li L 2019 A Service-oriented non-intrusive software fault-tolerant programming model WSEAS Transactions on Computer Research 7 46-50
[4] James B, Wirthlin M and Goeders J 2021 Investigating How Software Characteristics Impact the Effectiveness of Automated Software Fault Tolerance IEEE Transactions on Nuclear Science 68(5) 1014-22 DOI: 10.1109/TNS.2021.3073259
[5] Tyurin S F and Zarubskiy V G 2019 Automated control systems of increased reliability based on the use of configurable functionally complete tolerant logic elements WSEAS Transactions on Systems 18 282-8
[6] Subasi N, Guner U and Ustoglu I 2021 N-version programming approach with implicit safety guarantee for complex dynamic system stabilization applications Measurement and Control (United Kingdom) 54(3-4) 269-78 DOI: 10.1177/0020294019887473
[7] Machida F 2019 N-version machine learning models for safety critical systems Proc. 49th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks Workshop, DSN-W 2019
8. Avižienis A 1985 The N-Version Approach to Fault-Tolerant Software IEEE Transactions on Software Engineering SE-11(12) 1491-501
9. Chen L and Avižienis A 1978 N-version programming: a fault-tolerance approach to reliability of software operation Proc. Annual Int. Conf. on Fault-Tolerant Computing 3-9
10. Golubev I M, Tsarev R Ju and Semenko T I 2005 N-version software systems design Proc. 11th Int. Scientific and Practical Conf. of Students Postgraduates and Young Scientists; "Modern Techniques and Technologies" MTT 2005 (Tomsk, Russia: IEEE) 147-9
11. Girdhar V and Al-Masri E 2020 N-Version Programming for Enhancing Fault Tolerance in Fog-based IoT Systems Proc. 6th Int. Conf. on Science in Information Technology: Embracing Industry 4.0: Towards Innovation in Disaster Management ICSITech 2020 (Palu, Indonesia: IEEE) 109-14 DOI: 10.1109/ICSITech49800.2020.9392033
12. Gupta R and Jain M 2020 Reliability of N-version programming software with testing effort International Journal of Reliability and Safety 14(4) 229-51 DOI: 10.1504/IJRS.2020.114694
13. Gruzenkin D V, Yakimov I A, Kuznetsov A S, Tsarev R Yu, Grishina G V, Pupkov A N and Bystrova N V 2019 Algorithm diversity metric for N-version software Journal of Physics: Conference Series 1333(3) 032086
14. Tsarev R Y, Durmuş M S, Üstoglu I and Morozov V A 2018 Classification of voting algorithms for N-version software Journal of Physics: Conference Series 1015(4) 042060
15. Tsarev R Y, Durmuş M S, Üstoglu I and Morozov V A 2018 Application of majority voting and consensus voting algorithms in N-version software Journal of Physics: Conference Series 1015(4) 042059
16. Tsarev R Y, Durmuş M S, Üstoglu I, Morozov V A and Pupkov A N 2019 Fuzzy voting algorithms for N-version software Journal of Physics: Conference Series 1333(3) 032087
17. Gruzenkin D V, Tsarev R Y and Pupkov A N 2016 Technique of selecting multiversion software system structure with minimum simultaneous module version usage Advances in Intelligent Systems and Computing 465 375-86
18. Tsarev R Y, Gruzenkin D V and Grishina G V 2019 Recursive algorithm for exhaustive search of possible multiversion software realizations with the choice of the optimal versions set Advances in Intelligent Systems and Computing 763 30-6
19. Gruzenkin D V, Grishina G V, Durmuş M S, Üstoglu I and Tsarev R Y 2017 Compensation model of multi-attribute decision making and its application to N-version software choice Advances in Intelligent Systems and Computing 575 148-57
20. Kovalev I, Zelenkov P, Karaseva M and Kovalev D 2015 Ant algorithm modification for multiversion software building Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 9140 222-8
21. Saramud M V, Kovalev I V, Losev V V, Karaseva M V and Kovalev D I 2018 On the application of a modified ant algorithm to optimize the structure of a multiversion software package Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10941 91-100
22. Saramud M V, Kovalev I V, Losev V V and Voroshilova A A 2019 Multiple start modifications of ant colony algorithm for multiversion software design Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11655 191-201
23. Dorigo M 1992 Optimization Learning and Natural Algorithms (Milano, Italy: Politecnico di Milano)
24. Dorigo M and Stutzle T 2004 Ant Colony Optimization (Massachusetts , US: Massachusetts Institute of Technology)
25. Wang Y and Han Z 2021 Ant colony optimization for traveling salesman problem based on parameters optimization Applied Soft Computing 107 107439