УДК 004.052; 004.416.2
DOI: 10.31891/CSIT-2021-5-9

VITALIY YAKOVYNA, IVAN SYMETS
Lviv Polytechnic National University

A METHOD OF HIGH-ORDER MARKOV CHAIN REPRESENTATION THROUGH AN EQUIVALENT FIRST-ORDER CHAIN FOR SOFTWARE RELIABILITY ASSESSMENT

Modern complex technical systems, including but not limited to embedded, IoT and telecommunication systems are computer devices in which software plays a significant role. Wide use of such systems for critical applications in terms of failures increases the requirements for reliability and safety of both such systems in general and their software component. At the same time, the complexity of such systems is constantly increasing. To increase the certainty and accuracy of assessing the reliability of modern complex technical systems, it is necessary to use reliability models with a high degree of adequacy. Among the software reliability models, architectural models based on high-order continuous time Markov chains have a high degree of adequacy. How-ever, the practical use of such models for systems with many states is com-plicated due to the lack of practical methods and algorithms for calculating the characteristics of such systems. This paper solves the problem by presenting such models in the form of equivalent first-order Markov processes. The paper describes a method for representing a high-order Markov process in the form of an equivalent first-order process with additional virtual states. The proposed approach makes it possible to integrate high-order reliability models into the existing software tools for the analysis of reliability indices of complex technical systems.

Keywords: Software Dependability, High-Order Markov Chain, Software Reliability Model, Method, Equivalent Process.

ВІТАЛІЙ ЯКОВИНА, ІВАН СИМЕЦЬ
Національний університет «Львівська політехніка»

МЕТОД ПРЕДСТАВЛЕННЯ МАРКОВСЬКОГО ПРОЦЕСУ ВИЩОГО ПОРЯДКУ У ВИГЛЯДІ ЕКВІВАЛЕНТНОГО ПРОЦЕСУ ПЕРШОГО ПОРЯДКУ ДЛЯ ОЦІНЮАННЯ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Сучасні складні технічні системи, включаючи вбудовані, IoT і телекомунікаційні системи є програмно-апаратними пристроями, в яких програмне забезпечення відіграє значну роль. Широке використання таких систем для критичних додатків з точки зору відмов підвищує вимоги до надійності та безпеки як цих систем загалом, так і їх програмної складової. При цьому складність таких систем постійно зростає. Для підвищення визначеності та точності оцінки надійності сучасних складних технічних систем необхідно використовувати моделі надійності з високим ступенем адекватності. Серед моделей надійності програмного забезпечення високим ступенем адекватності володіють архітектурні моделі, засновані на ланцюгах Маркова вищого порядку з неперервним часом. Однак практичне використання таких моделей для систем з багатьма станами ускладнене через відсутність практичних методів і алгоритмів розрахунку характеристик таких систем.

Ця стаття вирішує проблему шляхом подання таких моделей у вигляді еквівалентних Марковських процесів першого порядку. У статті описується метод представлення Марковського процесу вищого порядку у вигляді еквівалентного процесу першого порядку з додатковими віртуальними станами. Запропонований підхід дає змогу інтегрувати моделі надійності вищого порядку в існуючі програмні засоби для аналізу показників надійності складних технічних систем. Цей метод заснований на методах обходу графів і схожий на алгоритм Дейкстри. Розглянутий метод можна застосувати до Марковських процесів змінного порядку, використовуючи різні значення змінної, що відповідає порядку процесу для кожного стану.

У статті також наведено приклад використання розробленого методу, що підтверджує практичну цінність та ефективність реалізації методу, а також дозволяє інтегрувати моделі надійності вищого порядку в існуюче програмне забезпечення для аналізу надійності складних технічних систем. Верифікація методу здійснена шляхом порівняння результаті його роботи з еквівалентним графом, побудованим вручну.

Ключові слова: надійність програмного забезпечення, ланцюг Маркова вищого порядку, модель надійності програмного забезпечення, метод, еквівалентний процес.

Introduction

Most modern technical equipment, in particular – electronic and telecommunication devices, are computer systems, the functioning of which includes an interaction of software and hardware. The assessment and ensuring of the specified reliability indices due to their increasingly responsible functions related to human safety is of particular importance for their design, production, and operation. The need for reliable software and the concept of software reliability engineering to reach this need was introduced by M. Lyu in his “Handbook of Software Reliability Engineering” [1]. Besides, there is a steady tendency to increase both structural and functional complexity of software and hardware systems. With the in-creasing complexity of computer systems, the procedure of certain and accurate assessment of their reliability indices becomes more complicated. In case of development of such systems, a significant part of the budget – up to 29% according to Capgemini data (https://bit.ly/36AwHfO) – and of the project implementation time is allocated for analysis and ensuring the intended requirements for reliability and safety. Thus, the current stage of computer systems development is characterized by a clear contradiction between the responsibility and complexity of the respective software – on the one hand, and methods and means of its reliability assessment and prediction – on the other hand. This contradiction can be eliminated by increasing the degree of adequacy of software reliability models, as well as considering the impact of architecture and software complexity on the evaluation of its

reliability indices.

During the recent decades, a significant number of analytical models has been proposed and investigated to assess the quality and reliability of software. Each model is based on certain assumptions about the software development process and operating environment. The operating environment differs from the program testing environment, and may vary for each software, life cycle development process, as well as depending on the technical capabilities of the project team [2].

Due to the growing software complexity the 'black box' models, including the most used nonhomogeneous Poisson process (NHPP) models, ceased to describe the software reliability behavior with a sufficient accuracy. Therefore, since the 90s of the last century, the 'white box' models, which describe software reliability considering its internal structure, are becoming more widespread. In models of this type, the reliability of the system is a function of the reliability of its modules. Such models have become especially popular in the last decade, as most modern complex multicomponent software products are based on an object-oriented paradigm, i.e., when the program is considered as a set of parallel entities (objects) interacting with each other. At present, several approaches have been developed to solve this problem, and models for assessment of the reliability of a multicomponent software system have been proposed [3–16]. In their turn, component-based models are divided into additive models; execution path models and on architecture models [3, 4]. Thus, one of the ways to increase the degree of adequacy of software reliability models is to develop and improve component-based software reliability models.

A significant part of component models are models based on the architectural approach [3, 4, 7, 9, 10, 11, 14]. In models of this type, a control flow graph is used to describe the software architecture. It is believed that the transfer of control between modules is characterized by Markov property, i.e., the future behavior of the system does not depend on its functioning in the past but depends only on the current state (being in a certain state means the execution of the respective software module). In this approach, the software architecture can be modeled as a discrete or continuous time Markov chain or as semi-Markov process. Each of these models can be absorbing (containing an absorbing state, i.e., a state from which the system cannot exit) and non-absorbing (without absorbing states). Besides, these models can be divided into compositional and hierarchical [10]. Compositional models include models that simultaneously combine the software architecture and the nature of its failures to determine the software reliability. In hierarchical models, the parameters of the architectural model are first calculated, and then the behavior of system failures is considered when predicting the software reliability.

One of the significant disadvantages of the 'white box' models is the complexity of their practical use in the software industry. Many parameters in these models are needed to be known in advance, but there are no described ways to obtain their values for real software. At the same time, software reliability assessment models based on an architectural approach, in most cases, use the theory of classic Markov processes, assuming the independence of the software system modules, which is a simplified description of the actual software execution process. To increase the degree of adequacy of such models, and, respectively, to increase the certainty of reliability assessment of complex software systems, it is necessary to consider that the probability of transition to the next state (for example, the transfer of a control flow to another module of the program) may depend not only on the current state but also on the background of this state (i.e., which program module had been executed before the current module). In [12–14], to avoid simplification, it is proposed to use high-order Markov chains (HOMC), which allow assessing with more accuracy the software reliability, as the interdependence of execution of their modules is considered.

Related works

In the previous works, one of the authors of this article developed and studied a software reliability continuous time high-order Markov chains model [14, 15], which consists of the following components: $\{C_i\}$ – directional graph that reflects the software structure, the vertices of the graph correspond to the software modules, and the edges reflect the control flow between modules ($i = 1, N$, where N – number of modules of the software system); $A = \{a_{ij}(t)\}$ – the transition intensity matrix of the system; $\boldsymbol{P} = \{p_i(t)\}$ the probability vector of the system being in the state C_i at time t; $\lambda_i\,(t)$ – failure rate of i-th software module. We should point out that the matrix of the intensity of transitions between states of the system reflects the number of transitions from the state *i* to the state *j* per unit of time, and in general its elements may be time-dependent $a_{ij} = f(t)$, and in case of HOMC, depend on the way the system has reach the state *i*.

In this model, the failure rate of a software system consisting of *N* modules can be written as:

$$\lambda(t) = \sum_{i=0}^{N} p_i(t)\lambda_i\,(t), \tag{1}$$

here $\lambda_i\,(t)$ – failure rate of i-th component, $p_i(t)$ – probability of execution of *i*-th software module at time *t* (which corresponds to the being of the system in the state C_i).

If the control flow between the modules of the system is modeled as continuous time Markov process (the state C_i of the process corresponds to the execution of *i*-th software module), the probability of the system being in the *i*-th state, $p_i(t)$ is obtained from the solution of Kolmogorov–Chapman equation system:

$$\frac{dp_i(t)}{dt} = -\sum_{j\in s} a_{ij}(t)p_i(t) + \sum_{j\in s} a_{ji}(t)p_j(t), i \in S, \tag{2}$$

here $a_{ij}(t)$ is the transition intensity from the state *i* to the state *j*, and *S* denotes the set of all system states.

In [16, 17], the influence of the process order is expressed as the dependence of the transition intensity on

the history of reaching the current state, and the higher is the order of the process, the longer the chain of transitions affects this intensity. In this case, the system of equations (2) must be modified accordingly. Another approach to consider the high-order process was proposed in [15], where the authors present a method for constructing an equivalent first-order process, the use of which reduces the problem to calculating the probabilities $p_i(t)$ based on the classic system of Kolmogorov–Chapman equations (2).

The goal of this work, which continues the research described in [14, 15], is to develop a method for constructing an equivalent graph of the states of a software system for high-order Markov process.

It is known from the theory of Markov processes that a high-order process can be represented as a first-order process by corresponding redefinition of the state space. For practical usage of HOMC models, it is necessary to have a formalized method for such a representation. Thus, for example, in [16, 17] a matrix of probabilities of the Markov process of the second order P^2, containing *S* states, is proposed to be represented in the form:

$$p^2 = \begin{pmatrix} p_{111} & \cdots & p_{11N} \\ \vdots & \ddots & \vdots \\ p_{NN1} & \cdots & p_{NNN} \end{pmatrix}$$

here p_{ijk} is the probability of transition from the state C_i to the state C_j, and then to the state C_k. However, such a matrix is very sparse, since there are no second-order paths between the respective states, so this representation is not effective from the point of view of software implementation, as it requires much memory to store zero elements of the matrix or use of specialized methods and data structures to work with sparse matrices.

Instead, in [14, 15], a high-order Markov process is presented in the form of an equivalent first-order process with additional virtual states. In this case, each state of the initial graph is split into as many virtual states as there are different paths to this state.

To illustrate the approach proposed in [15] the transition graph for some system of four modules is shown in Fig. 1(a). If we consider the second order Markov process, we can reach the state C_1 by two ways $C_1 \rightarrow C_2 \rightarrow C_4$ and $C_3 \rightarrow C_2 \rightarrow C_4$. Respectively, the state C_2 can be split into two virtual states C_2^1 and C_2^2 and the system will have the form presented in Fig. 1(b).

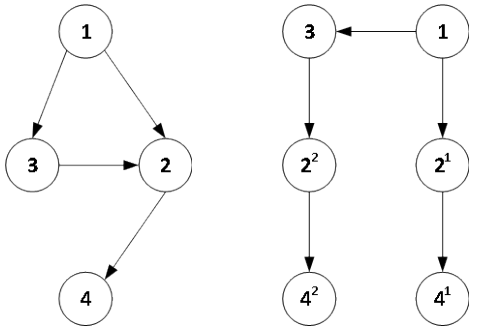

Fig. 1. An example of the equivalent transformation of the second order Markov process into the traditional first-order process (a – graph of states of the original system, b – graph of states of the system with virtual states)

Let M^k be a matrix the elements of which m_{ij}^K denote the number of possible paths of the order *K* from the state C_i to C_j. Let us consider the matrix M^1, the elements of which are the number of first-order transitions between the corresponding states shown in Fig. 1(a) ($m_{12}^1 = 1$ is the element of the matrix M^1, which means that from the state C_1 to C_2 there is one possible first-order path):

$$M^1 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

It is easy to show that the matrix M^2, which will contain the number of chains of less or equal to the second order from the state C_i to C_j, will look like:

$$M^2 = \begin{pmatrix} 0 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

here $m_{12}^2 = 2$ means that from the state C_1 to the state C_2 there are two paths – one of the length equal to two (second-order Markov chain) $C_1 \rightarrow C_3 \rightarrow C_2$ and one chain with the unity length: $C_1 \rightarrow C_2$.

The matrix M^k has the following property: the sum of the elements in *j* column of the matrix is equal to the number of paths entering the state C_j (the number of virtual states to which one needs to split the state C_j), and the sum of the elements in *i* line of the matrix is equal to the number of paths coming out of the state C_i.

Thus, to calculate the number of K-order chains from the state C_i to the state C_j), we can use a formula based

on the Floyd method:

$$m_{ij}^{K} = \sum_{j=1}^{N}(m_{il}^{K-1} + e_{il})m_{lj}^{1}, \quad (3)$$

here *N* is the number of all states of the system; e_{ij} denotes the elements of the identity matrix.

Based on (3), we can get an equivalent graph of the states of the system $\{C_i\}$ which is equivalent to the original graph $\{C_i\}$ in case of K order process. This allows us to construct a system of Kolmogorov–Chapman equations (2) for an equivalent graph and, from its solution, to obtain the probability of the system being in the states C_i, which corresponds to the probability of execution of *i*-th software module at time *t*. Having the values of $p_j(t)$ from the Eq. (3) one can calculate the reliability of the system using the Eq. (2). In case of variable order Markov process, a different value of the order of the process should be used for each state in the Eq. (3).

Method of High-Order Markov Chain Representation through an Equivalent First-Order Chain for Software Reliability Assessment

The input data for the developed method are the initial graph, which shows the modules and interaction flows between them (Fig. 2) and the order of the Markov process *N*.

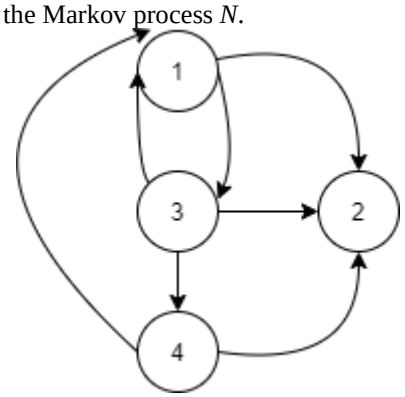

Fig. 2. An example of states and transitions graph

We will represent the graph in the form of an adjacency matrix, which is one of the convenient ways to represent a graph:

$$\begin{pmatrix} r_{11} & \cdots & r_{1m} \\ \vdots & \ddots & \vdots \\ r_{m1} & \cdots & r_{mm} \end{pmatrix},$$

here r_{ij} is a binary value, where “1” stands for existing an edge that connects the vertex M_i and M_j, while “0” denotes the absence of such connection; m is the number of modules in the system.

This method is similar in behavior to Dijkstra's algorithm, but its purpose is not to find a shorter path from a given vertex, but to find all possible paths of length N - 1 for each vertex of the input graph.

The main task of the method is to determine the states and transitions between them, which consider the N-1 previous states for the system under study. The method consists of two main steps

First, all the vertices of the input graph are sequentially processed and the list of vertices which lead to i-th vertex is determined using the adjacency matrix. In the next iteration (incremented order), each vertex from the built during the previous iteration list of vertices is considered and the updated list of vertices leading to i-th vertex is determined. This process is repeated N-1 times until we reach the desired order of the Markov chain.

As a result of this step, the list of k tree structures is obtained, where k is the number of nodes in the input graph. Each tree structure contains a node from the initial graph as a root node. For each of the nodes of the tree, the ancestors are the nodes which are previous states according to the adjacency matrix and the depth of each tree is N-1.

The next step is to process and run through the list of trees obtained during the first step. The purpose of this step is to determine the states of the equivalent graph for the N-order Markov chain. To determine these states, we need to go through all the branches of the trees (the path from the root node to each of the external nodes of the tree). The set of the corresponding unique paths will be the set of states of the equivalent graph for the N-order Markov chain.

Method Description

Notation:

N is the order of the graph to be determined using the method.

k is the number of nodes of the input graph.

inputGraphNodes is the list of vertices of the input graph.

inputNodeId is node id from input graph

adjacencyMatrix[*k*][*k*] is the adjacency matrix for the input graph.

listOfTreeNodes is a list of all nodes for a list of k (number of nodes in the input graph) tree structures, each of which will contain a node from the initial graph as the root node.

treeLevel the level of the tree.
nodeChildrenList is the list of child nodes for the tree node in the TreeNode structure.
HOMC_GraphNodes is the list of nodes of the equivalent graph for the *N*-order Markov chain.

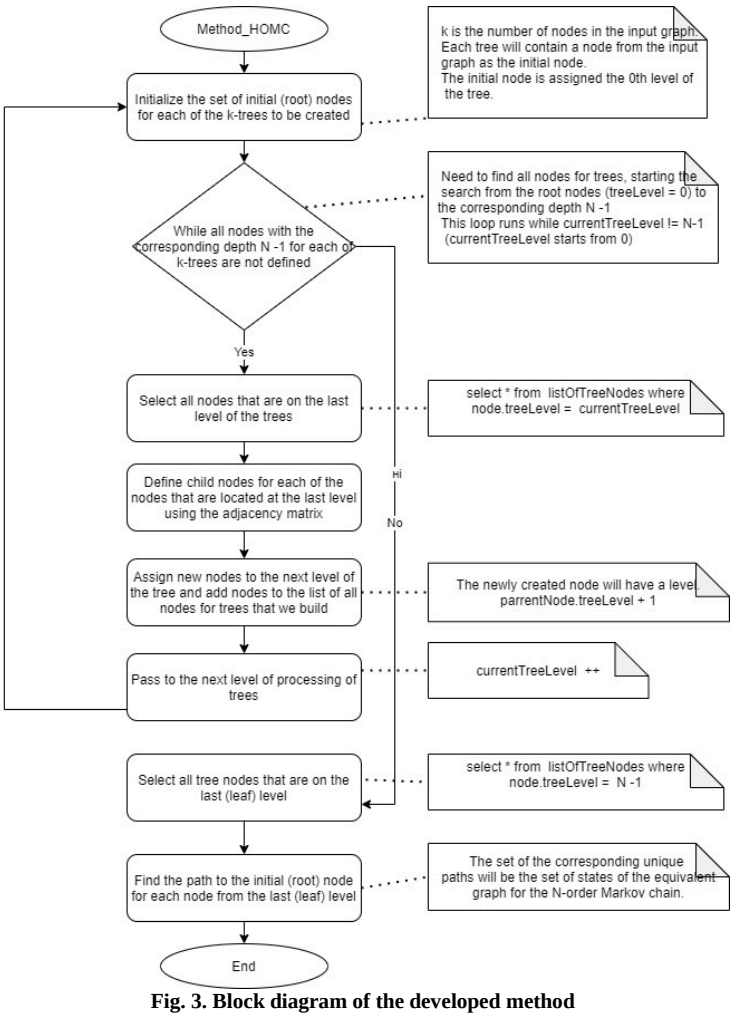

Fig. 3. Block diagram of the developed method

The pseudocode for the proposed method of high-order Markov chain representation through an equivalent first-order chain is given below.

```
function Method_HOMC(inputGraphNodes,adjacencyMatrix, N):

foreach inputGraphNode in inputGraphNodes:
newTreeNode ← inputGraphNode
newTreeNode.treeLevel ← 0
  add newTreeNode to listOfTreeNodes
end foreach

// filling the listOfTreeNodes
currentTreeLevel  ←  0
while currentTreeLevel is not equal N-1:
foreach node in (select * from listOfTreeNodes where treeLevel = currentTreeLevel):
  // find nodeChildrenList for node
  for indexI  ←  0 to indexI < k
 if adjacencyMatrix[indexI][node.inputNodeId]=1
     then
                  newTreeNode ← inputGraphNodes[node.initialNodeId]
   newTreeNode.treeLevel <- currentTreeLevel + 1
           add newTreeNode to treeNode. nodeChildrenList
               add newTreeNode to listOfTreeNodes
    end if
```

```
          end for
        end foreach
        currentTreeLevel ++
      end while

      foreach leafNode in (select * from listOfTreeNodes where treeLevel = N - 1):
         HOMC_GraphNodes ← add the path from the leafNode node to the tree (root)
node // this path is one of the states of the equivalent graph for the Markov chain
of the nth order

      return HOMC_GraphNodes
```

An example of using the develop method

Consider initial graph depicted in Fig. 2 that shows the modules and interaction flows between them for a test system, let us determine the equivalent graph for the second-order Markov chain. The adjacency matrix for this graph is shown in Fig. 4.

i\j	M_1	M_2	M_3	M_4
M_1	1	1	1	0
M_2	0	0	0	0
M_3	1	1	0	1
M_4	1	1	0	0

Fig. 4. Adjacency matrix for the graph represented in Fig. 2.

Step 1. Go through the graph and determine the list of vertices which lead to each of the vertices. Repeat this process *N-1* times, as we have *N=2*, we need to perform the first iteration.

Vertex #1. As can be seen in the adjacency matrix, vertex #1 has the transitions from the vertices #1 (loop), #3 and #4.

Vertex #2. Vertex #2 can be reached from the vertices #1, #3, and #4.

Vertex #3. Vertex #3 can be reached only from the vertex #1. Respectively, the tree for vertex #3 contains only one branch.

Vertex #4. As can be seen in the adjacency matrix, vertex #4 has the transition only from the vertex #3.

Thus, after completing this step, we have four tree structures, which reflect the parent and child vertices as the states of the system and the states which generate it.

Step 2. In this step, we need to determine the set of states of the equivalent graph for the second-order Markov chain. To do this, we need to go through all the branches of the formed trees (see Fig. 5) and determine the states that will be equal to the path leading from the root node to each of the leaves.

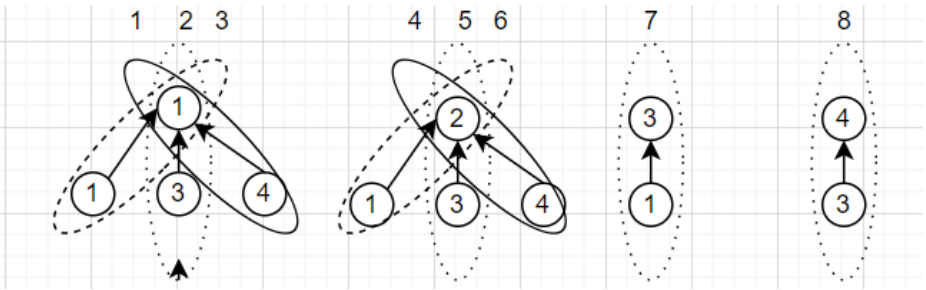

Fig. 5. List of the trees for each vertex and identification of tree branches

As a result, we have formed the following states of the equivalent graph for the second-order Markov chain and listed them in Table 1.

Table 1.

States of the equivalent graph for the second-order Markov chain for initial graph depicted in Fig. 2.

Transition in the initial graph	Resulting state in the equivalent graph
1 → 1	1_1
3 → 1	1_3
4 → 1	1_4
1 →2	2_1
3 → 2	2_3

$4 \rightarrow 2$	2_4
$1 \rightarrow 3$	3_1
$3 \rightarrow 4$	3_4

According to the set of states, the equivalent graph for the second-order Markov chain will look like it is shown in Fig. 6.

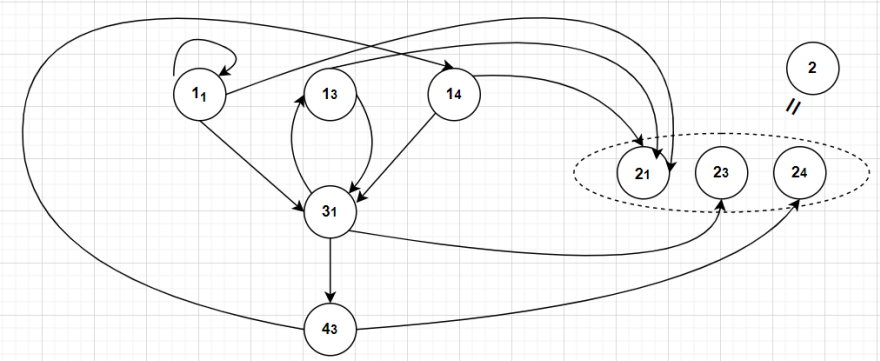

Fig. 6 The equivalent graph for the second-order Markov chain

To demonstrate the practical usage and to verify the developed method it was implemented and tested on an example of CubeSat nanosatellites flight software reliability assessment [18]. The simulations [18] show that high-order Markov chains can increase the accuracy of the failure rate assessment up to 50% comparing to the traditional Markov chain model, while the difference between the results obtained using the second and third-order models is not so significant (less than 10%). Thus, when assessing the reliability of modern software systems, high-order processes, i.e., the interdependence of software modules, should be considered. To do this, HOMC reliability models can be applied using the described method.

Conclutions and Future Work

Growing operational responsibility of modern computer systems, including control, IoT, telecommunication etc., leads to increasing requirements to its reliability. Reliability assessment of such systems is impossible without software reliability consideration. Growing operational responsibility of modern computer systems results in their increasing complexity. Increasing software complexity could make traditional software reliability models obsolete. To increase the certainty and accuracy of software reliability assessment high-order Markov chains can be used. However, complex mathematical description of such models prevents their practical usage.

This paper presents a method which transforms a high-order Markov chain, including variable-order one, to equivalent first-order Markov chain with additional virtual states, thus make it possible to use traditional methods and tools to deal with it. This method is based on graph traversal methods and is like Dijkstra's algorithm. The considered method can be applied to variable-order Markov processes by using different values of the variable corresponding to the order of the process for each state.

The article also provides a usage example of the developed method, which confirms the ability and efficiency of implementation of the method and allows the integration of high-order reliability models into existing software for reliability analysis of complex technical systems, like the developed by the authors of this article software suite for reliability assessment of complex technical systems [19]. The method is verified by comparison of its output with manually built equivalent graph.

High-order reliability models can be used in a variety of complex software and hardware systems [20]. Future work will be devoted to the study of the impact of high-order reliability models on the accuracy of reliability indices assessment for both pure software and mixed software-and-hardware systems.

References

1. Lyu, M.R.: Handbook of Software Reliability Engineering. IEEE Computer Society Press and McGraw-Hill Book Company, New York (1996).
2. Malaiya, Y.K., Srimani, P.K.: Software Reliability Models: Theoretical Developments, Evaluation and Applications. IEEE Computer Society Press, Los Angeles (1990).
3. Gokhale, S.S., Wong, W.E., Horgan, J.R., Trivedi, K.S.: An Analytical Approach to Architecture-Based Software Performance and Reliability Prediction. Performance Evaluation 58(4), 391–412 (2004).
4. Goševa-Popstojanova, K., Trivedi, K.S.: Architecture-based approach to reliability assessment of software systems. Performance Evaluation 45, 179–204 (2001).
5. Palviainen, M., Evesti, A., Ovaska, E.: The reliability estimation, prediction and measuring of component-based software. The Journal of Systems and Software 84, 1054–1070 (2011).
6. Shooman, M.L.: Structure models for software reliability prediction. In: Proceedings of the 2nd International Conference on Software Engineering, pp. 268–280. IEEE Computer Society Press, Washington, DC, USA (1976).
7. Xie, M., Wohlin, C.: An additive reliability model for the analysis of modular software failure data. In: Proceedings of the 6th International Symposium on Software Reliability Engineering, pp. 188–194. IEEE, Toulouse, France (1995).
8. Krishnamurthy, S., Mathur, A.P.: On the estimation of reliability of a software system using reliabilities of its components.

In: Proceedings of the 8th International Symposium on Software Reliability Engineering, pp. 146–155. IEEE, Albuquerque, NM, USA (1997).

9. Goševa -Popstojanova, K., Hamill, M.: Architecture-Based Software Reliability: Why Only a Few Parameters Matter? In: 31st Annual International Computer Software and Applications Conference, pp. 423–430. IEEE, Beijing, China (2007).
10. Gokhale, S., Trivedi, K.: Structure-based software reliability prediction. In: Proceedings of the Fifth International Conference on Advanced Computing, pp. 447–452. Association for Computing Machinery, New York, NY, USA (1997).
11. Gokhale, S., Wong, W.E., Trivedi, K., Horgan, J.R.: An analytical approach to architecture based software reliability prediction. In: Proceedings of the Third International Computer Performance and Dependability Symposium, pp. 13–22. IEEE, Durham, NC, USA (1998).
12. von Bochmann, G., Jourdan, G.-V., Wan, B.: Improved Usage Model for Web Application Reliability Testing. In: Wolff, B., Zaidi, F. (eds.) ICTSS 2011, LNCS, vol. 7019, pp. 15–31. Springer, Heidelberg (2011).
13. Takagi, T., Furukawa, Z., Yamasaki, T.: Accurate Usage Model Construction Using High-Order Markov Chains. In: Supplementary Proceedings of 17th International Symposium on Software Reliability Engineering, pp. 1–2. IEEE, Raleigh, NC, USA (2006).
14. Yakovyna, V., Nytrebych, O.: Discrete and Continuous Time High-Order Markov Models for Software Reliability Assessment. In: Proceedings of the 11th International Conference on ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer, pp. 419–431. CEUR-WS.org, Vol. 1356 (2015) online.
15. Yakovyna, V., Nytrebych, O., Fedasyuk, D.: The representation of high order Markov process through equivalent first order process. In: Proceedings of 6th International Conference of Young Scientists Computer Science and Engineering, pp. 216–217. Lviv Polytechnic Publishing, Lviv, Ukraine (2013).
16. Berchtold, A., Raftery, A.E.: The mixture transition distribution model for high-order Markov chains and non-Gaussian time series. Statistical Science 17(3), 328–356 (2002).
17. Shamshad, A., Bawadi, M.A., Wan Hussin, W.M.A., Majid, T.A.: First and second order Markov chain models for synthetic generation of wind speed time series. S.A.M. Sanusi Energy 30(5), 693–708 (2005).
18. Yakovyna, V., Symets, I.: Reliability assessment of CubeSat nanosatellites flight software by high-order Markov chains. Procedia Computer Science 192C, 447–456 (2021).
19. Yakovyna, V., Seniv, M., Symets, I., Sambir, N.: Algorithms and software suite for reliability assessment of complex technical systems. Radio Electronics, Computer Science, Control 4, 163–177 (2020).
20. Mulyak, A., Yakovyna, V., Volochiy B.: Influence of software reliability models on reliability measures of software and hardware systems. EasternEuropean Journal of Enterprise Technologies 4, 53–57 (2015).