Issue 16, 2024

https://doi.org/10.23939/sisn2024.16.290

UDC 004.05

IMPROVED SOFTWARE SYSTEM FOR CALCULATING THE RELIABILITY INDICATORS OF COMPLEX TECHNICAL SYSTEMS

Maksym Seniv[1], Stepan Zdebskyi[2]

Lviv Polytechnic National University,
Software Department, Lviv, Ukraine
[1] E-mail: maksym.m.seniv@lpnu.ua, ORCID: 0000-0003-1044-4628
[2] E-mail: stepan.zdebskyi.mnpzm.2024@lpnu.ua, ORCID: 0009-0000-4454-2539

The article analyses the literature sources, which investigate the existing methods and means of calculating reliability indicators of complex technical (in particular, software) systems. The reliability model of a modern complex technical system is often depicted in the form of a reliability block diagram (RBD), which may contain thousands of elements, each transitioning between different states (e.g., operational, failed, restored). This leads to a vast space of possible states in the corresponding Markov model. The reliability behaviour of a system is commonly described by a graph, with nodes representing system states and edges representing possible transitions between these states. A number of software products can be used to automate the calculation of reliability indicators for complex technical systems. However, these products have several limitations, including: difficulty in implementing into design and development processes; significant costs for licenses and staff training; lack of compatibility with other reliability analysis and life cycle management products; lack of tools for working with databases, etc. Most of the outdated products are desktop applications with an insufficiently user-friendly graphical interface. The primary objective of this work is to develop an improved software system that includes the modification and implementation of a recursive algorithm for forming an operability condition and visualizing a circular graph of states/transitions. With the help of the system, it is possible to automate the construction of reliability flowcharts for complex technical, in particular, software systems, calculate the operability condition using an improved recursive algorithm and method for determining the operability condition, determine the system states and visualize them using an n-ary or circular graph. Additionally, the system offers tools for calculating reliability indicators: availability and downtime factors, time between failures, failure flow parameters, etc. The advanced software system enables automated calculation of reliability indicators for software systems of any complexity level and reduces the influence of the human factor in the process of reliability design.

Keywords: software, RBD, reliability design, states and transitions graph.

Introduction and problem statement

The relentless growth of society's dependence on software and hardware systems has led to increasing demands for their reliability and safety. Accordingly, the relevance and importance of reliability analysis as a tool for ensuring the sustainable operation of technical systems is undeniable. Unfortunately, traditional methods of reliability analysis are often unable to adequately address the challenges posed by a large number of parameters and their interactions within complex systems. This leads to the need to develop new approaches and tools that would allow reliability analysis to be performed efficiently and with high accuracy. The lack of timely and objective reliability analysis can lead to serious consequences, ranging from financial losses to threats to human life.

The rapid growth in the complexity of technical systems requires that reliability analysis and forecasting tools be developed at the same pace. They are required not only to process more data, take into account a wide range of factors, adapt to new hardware and software systems, but also to ensure high accuracy and reliability of the results. A number of software products are currently used to automate the calculation of reliability indicators for complex technical systems, such as PTC Windchill [1], RAM Commander [2], BlockSim [3] and Isograph [4]. However, these products have several limitations, including: difficulty in implementing into design and development processes; significant costs for licenses and staff training; lack of compatibility with other reliability analysis and life cycle management products; lack of tools for working with databases, etc. Most of the outdated products are desktop applications with an insufficiently user-friendly graphical interface. Such systems usually do not provide bug fixes or new functionality. More modern products can be either desktop or web-based, and are characterised by a user-friendly interface and regular release of new versions. Accordingly, there is a need to develop new and improve existing tools for calculating reliability indicators of complex technical systems that would eliminate the above shortcomings and enable automated calculation of these indicators, while ensuring acceptable accuracy and cost of work.

Analysis of recent research and publications

Software reliability modelling is an important component of the process for ensuring quality and predicting system behaviour under various operating conditions. The reliability model of a modern complex technical system can include thousands of elements that alternately exist in different states (e. g., operational, failed, recoverable) [5]. This leads to a vast space of possible states in the corresponding Markov model. The reliability behaviour of a system is commonly described by a graph, with nodes representing system states and edges representing possible transitions between these states [5].

In [6], the issue of reliability modelling for uncertain environments is addressed, and a new model is proposed that minimizes the number of assumptions and parameters and is intended for broader use compared to traditional models. The proposed model was compared with 21 existing models using two datasets, 15 criteria, and a multi-criteria decision-making method. As a result, it was proven that the NHPP SRM (Non-Homogeneous Poisson Process Software Reliability Model), which takes uncertain operating environments into account and minimizes assumptions, is suitable for general situations with fewer parameters, demonstrating good results when compared to existing models [6].

The publication [7] proposes a neuro-fuzzy hybrid method that combines a self-organizing map and fuzzy time series forecasting to create a reliability prediction model. The methodology includes a clustering algorithm to divide software failure data into intervals, which are then used to create fuzzy sets. These sets are used to establish fuzzy logical relationships and their groups, based on which reliability forecasting is performed. The proposed neuro-fuzzy hybrid approach to software reliability prediction demonstrated a significant improvement in accuracy compared to existing fuzzy time series-based models [7]. Experimental results confirm the effectiveness of this method, making it suitable for improving software quality. The flexibility and adaptability of the proposed approach allow it to be applied to different types of software and usage conditions [7].

Queue-based simulations using the “first in, first out” principle are also used to describe behaviour, model the error detection process, and assess software reliability [8]. In [9], a data-driven software reliability model using deep learning is proposed, while in [10–11], software reliability growth models are based on the hypothesis derived from the logistic model and take into account the effectiveness of defect correction.

All of the aforementioned models and methods can be used to analyse the reliability of technical (particularly software) systems. However, their implementation in existing software products for automating the calculation of reliability indicators for complex technical and software systems is problematic, as it requires significant resources for programming and verifying the results of applying such models to real products. Therefore, an urgent task is the improvement of existing software products for the automated calculation of technical system reliability indicators.

Formulation of the article objectives

The aim of this work is to develop an advanced software system by modifying and implementing a recursive algorithm for determining the operability condition, as well as tools for visualizing a circular graph of states/transitions. With the help of this improved system, it will be possible to automate the construction of reliability block diagrams (RBD) for technical systems, calculate the operability condition using the improved recursive algorithm, determine system states, and visualize data using an n-ary or circular graph. In addition, the system provides tools for calculating various reliability indicators, such as availability and downtime factors, mean time to failure, failure rate parameters, and others. To achieve this goal, it is necessary to: analyse the existing methods and tools for calculating reliability indicators of complex technical systems; formulate the requirements for the improved software tool; design and implement the advanced software system for calculating the reliability indicators of complex technical systems; and verify the obtained results.

Presentation of the main material

The primary task of this research is to extend the functionality of the existing product [12] designed for the automated calculation of reliability indicators for technical systems, addressing users' needs for rapid and objective reliability analysis using RBD, state graphs, and statistical forecasting methods. Accordingly, the following components were developed:

1. A module for determining the operability condition based on an improved recursive algorithm for structural diagram analysis.
2. A module for visualizing a circular state graph.

Together, the system should provide the following functionality:

1. Construction of a reliability block diagram for technical systems.
2. Determination of the system's operability condition using the method [12] for operability determination or an improved recursive traversal algorithm.
3. Construction of n-ary and circular state and transition graphs for repairable and non-repairable systems.
4. Determination of the probability distribution of the system's states.
5. Calculation of the system's reliability indicators based on the probability distribution of its states.

During the modification of system [12], a decision was made to adhere to the principle of non-interference with the original code to avoid errors and ensure more flexible integration of new modules and the overall system. The VIPER_RC module calculates the operability condition of a technical system using the improved recursive algorithm. The VIPER_CircleGraph module is a web application designed for the construction and visualization of a circular state graph of the system. Additionally, a server application based on the ASP.NET Web API framework was developed to enable interaction between the main system and the VIPER_CircleGraph web application.

The core algorithm of the VIPER_RC module is an improved recursive algorithm for calculating the operability condition of a technical system. The input for the algorithm is the structural diagram of the technical system, while the output is a string containing a Boolean expression that defines the conditions under which the system remains operational. The recursive algorithm processes the structural diagram as a directed graph with two types of nodes: a module and a connector node. The main idea of the algorithm is to use a combined graph traversal and recursive calls for each segment of the graph, which is a part of the structural diagram and is separated by two nodes.

A connector node in the structural diagram of the technical system does not necessarily indicate the beginning of two or more parallel branches of the directed graph; however, if two or more modules or graph segments are executed simultaneously, they must begin and end at a node. Each connector node is either the beginning or the end of a segment (Fig. 1), which may contain only modules, parallel segments, or be empty. Each segment contains a start and an end connector node. Since a given system may be part of a larger system, the structural diagram of this segment begins with a start connector node and ends with an end node. Therefore, for the recursive algorithm, it is essential to have data on the start and end nodes of all segments of the diagram.

The first part of the algorithm involves searching for the start and end nodes for all segments of the system's graph. The start node indicates the initial point of traversal for a segment, while the end node marks the point where recursion ends at the current level. If the current node is the final node of the system's graph and all parallel segments have been processed, the result is returned, and the algorithm is completed.

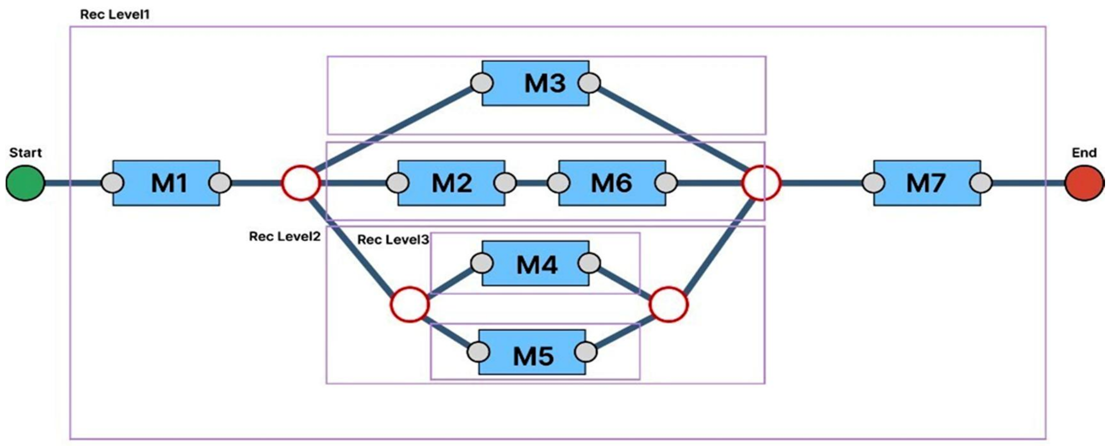

Fig. 1. Example of dividing a structural diagram into segments

One approach to this task is performing a depth-first search (DFS) on the structural diagram. Since each segment must have both a start and an end node, no segment can exist with only one or none of these nodes. By creating an array of all the diagram's connector nodes, we can use DFS to locate the corresponding end nodes. If the current search node is a start node of a segment, the end node is the next node at that level. Applying this algorithm to all connector nodes, except the system's final node, yields corresponding "start-end" pairs. According to this approach, a specific node can serve as both the start node for one segment and the end node for another. This behaviour is logical and permissible since a connector node can have multiple input and output connections, unlike a module, which should have only one input and one output connection. Finding the start and end for each segment in a scheme with solely sequential connections is a trivial task. However, the structural diagram can simultaneously contain both sequential and parallel connections. Therefore, the identified end node of a segment must be the next node at the current level. To achieve this, while searching for the end node of the current segment, DFS is performed from the current node to the final node of the entire diagram. The generated list of traversed nodes will certainly contain the end node for the current segment. Afterward, a similar traversal is performed through the other branch of the parallel connection. The first node found in both arrays is the end node of the current segment (Fig. 2).

Algorithm steps (Fig. 2):

1. Identify the starting node of the segment. The starting node for the search is the current node from the array of all nodes in the scheme.

2. If this node has more than one outgoing connection, choose two branches of the parallel connection. If this node has only one outgoing connection, perform a depth-first search to the end of the graph and designate the first node in the traversal array as the ending node, then proceed to step 5.

3. Select one branch of the segment and perform a depth-first search to the end of the graph. Record all visited nodes in a list.

4. Select the other branch of the segment and perform a depth-first search until a node that is in the list is found. If such a node is found, mark it as the end of the current segment.

5. The current element becomes the next element in the array of nodes in the structural scheme.

6. If all elements in the array of nodes in the structural are currently used to automate the calculation of reliability indicators for complex technical systems, such as PTC Windchill have been processed – end; otherwise, go to step 1.

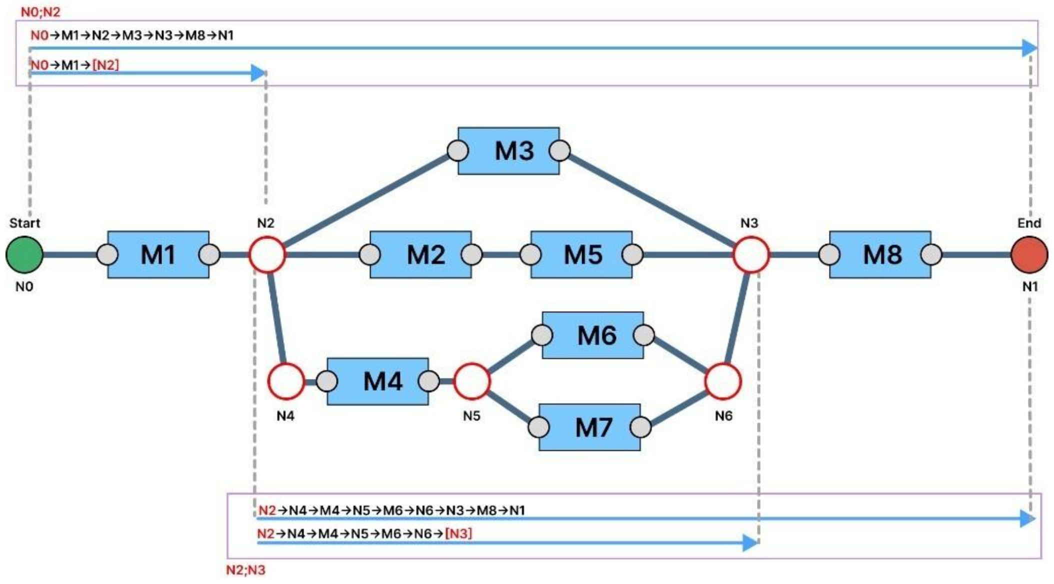

Fig. 2. Visualization of the algorithm for finding the starting and ending nodes for two segments of a diagram consisting of eight modules

The second part of the algorithm involves performing a combined recursive traversal of the graph. This part is the main component of the algorithm; it accepts a data structure describing the reliability scheme and a dictionary in which the key is the starting node of a certain segment, and the value is the end node. The core of the recursive algorithm is a function that executes when traversing the starting node of each segment and returns a value upon reaching its end. The returned value is a sub-expression that reflects the operational condition of the current segment. If the end node of the entire structural scheme is reached, the function will return a cumulative result obtained by combining all returned results.

Steps of aforementioned algorithm (Fig. 3):

1. Assign the current element of the search the value of the starting element of the segment. The current operational expression is empty.

2. Add the current element to the list of visited nodes. Assign the value of the end element for this segment using the dictionary of starting and ending nodes.

3. If the current element is the end element of the scheme, return the final result. If the current element is a module, add the expression “AND + [Name of the current module]” to the current formula, identify the current element as the next element of the scheme, and proceed to step 2. If the current element is a node, proceed to step 4.

4. Calculate the number of outgoing connections of the node. If the number of outgoing connections of this element is equal to one, then the current element becomes the next element of the scheme; proceed to step 2. If the number of outgoing connections is greater than one, proceed to step 5.

5. For each individual branch of the current branching, a recursive function call is performed, specifying as parameters the starting element as the next element for each branch and the end element obtained from the dictionary of starting and ending nodes for the current node. The list of visited elements of the scheme is also passed as an argument. The recursive call brings the algorithm back to step 1 for another function stack. Each returned value is added to the current operational expression according to the formula: "[current expression] AND ([branch expression 1] OR [branch expression 2] OR … OR [branch expression N])". If all paths have been explored, the current element is assigned the end node of the segment; proceed to step 2.

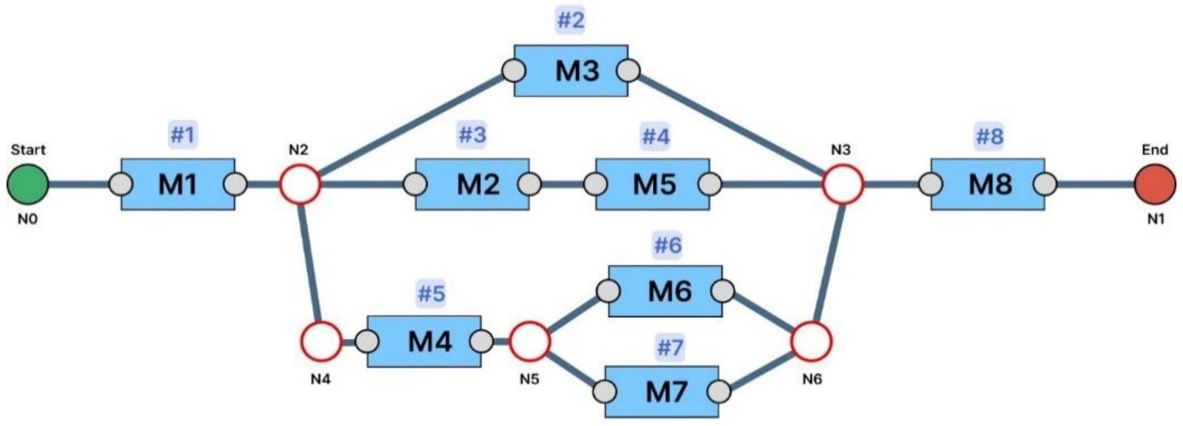

Fig. 3. Traversing order of the RBD using the improved recursive algorithm

At the top level of recursion, the result obtained from traversing all nodes and segments is the final output of the algorithm. It is crucial to consider the nuances of representing logical operations "OR" and "AND." For correctly constructing the expression, it is important to account for the possible arrangements of nodes and modules. This also includes parentheses, as their incorrect placement in the formula can distort the result. Additionally, certain segments may be empty, meaning they contain no modules. In such cases, these segments should not influence the overall operational expression. The block diagram representation of the improved recursive algorithm is shown in Figures 4 and 5.

The software implemented in this work consists of two components. The first part is an extension of the original WPF application and performs the calculation of the operational expression of the technical system using the improved recursive algorithm. This part was developed using C# and WPF tools on the .NET version 5.0.0 platform. The second part is a web application designed for constructing and visualizing the circular state graph of the system (Fig. 6). The implementation of the web application was carried out using the React.js framework. The libraries used include ReactFlow version 11.11.2, which contains modern tools for building graphs of any structure and complexity, ReactRouterDOM version 6.22.3 for facilitating quick navigation between the application pages, and D3 version 5.5.0 for interactive data visualization. The server application was implemented based on the ASP .NET Web API framework and is responsible for the interaction between the main system and the web application.

In the reliability analysis panel, selecting the "Circular Graph" option opens the main page of the web application displaying the circular state graph. The page contains the main working area where the circular state graph is displayed as a graph, with elements arranged in a circle (Fig. 6). Additionally, the page includes a navigation panel and a mini map. The user can double-click the left mouse button on a node of the graph, after which a page with detailed information about the graph will open

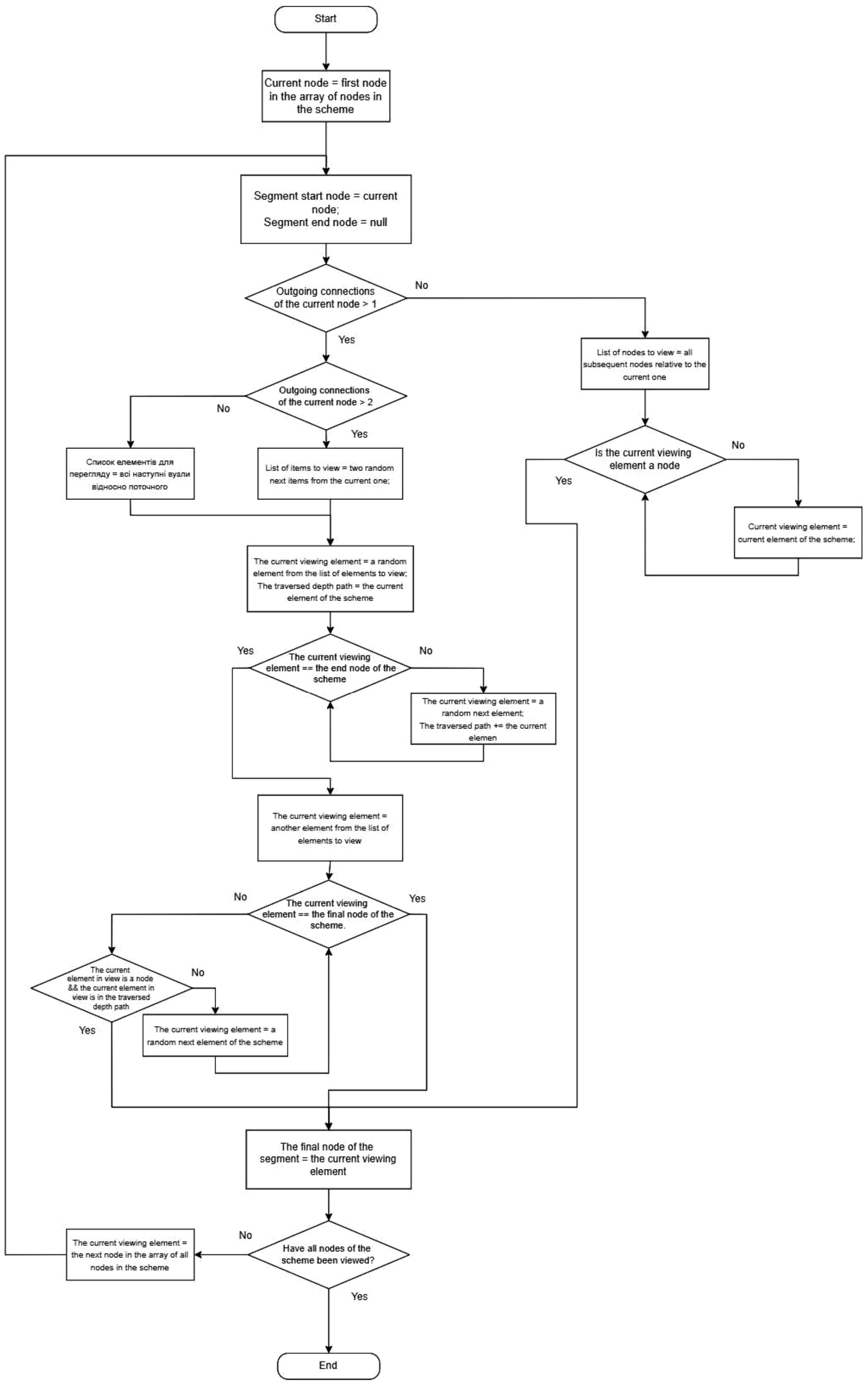
Start
Current node = first node in the array of nodes in the scheme
Segment start node = current node;
Segment end node = null
Outgoing connections of the current node > 1
No
Yes
List of nodes to view = all subsequent nodes relative to the current one
Is the current viewing element a node
No
Yes
Current viewing element = current element of the scheme;
Outgoing connections of the current node > 2
No
Yes
Список елементів для перегляду = всі наступні вузли відносно поточного
List of items to view = two random next items from the current one;
The current viewing element = a random element from the list of elements to view;
The traversed depth path = the current element of the scheme
Yes
No
The current viewing element == the end node of the scheme
The current viewing element = a random next element;
The traversed path += the current elemen
The current viewing element = another element from the list of elements to view
No
Yes
The current viewing element == the final node of the scheme.
The current element in view is a node && the current element in view is in the traversed depth path
No
Yes
The current viewing element = a random next element of the scheme
The final node of the segment = the current viewing element
No
Have all nodes of the scheme been viewed?
The current viewing element = the next node in the array of all nodes in the scheme
Yes
End

Fig. 4. Block diagram representation of the first part of the improved recursive algorithm – searching for segment boundaries

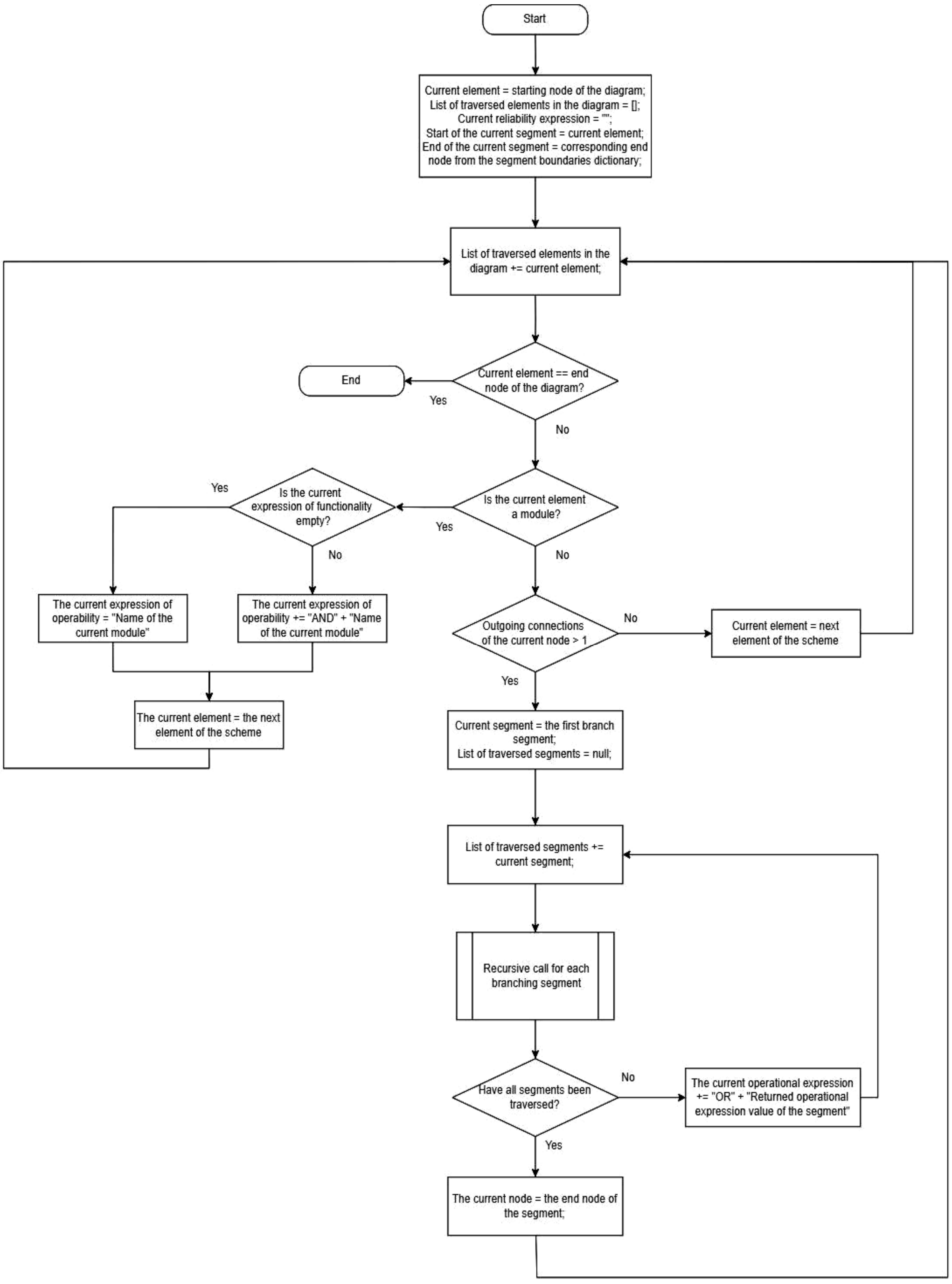
Start
Current element = starting node of the diagram;
List of traversed elements in the diagram = [];
Current reliability expression = "";
Start of the current segment = current element;
End of the current segment = corresponding end node from the segment boundaries dictionary;
List of traversed elements in the diagram += current element;
Current element == end node of the diagram?
Yes
End
No
Is the current element a module?
Yes
Is the current expression of functionality empty?
Yes
No
The current expression of operability = "Name of the current module"
The current expression of operability += "AND" + "Name of the current module"
The current element = the next element of the scheme
No
Outgoing connections of the current node > 1
No
Current element = next element of the scheme
Yes
Current segment = the first branch segment;
List of traversed segments = null;
List of traversed segments += current segment;
Recursive call for each branching segment
Have all segments been traversed?
No
The current operational expression += "OR" + "Returned operational expression value of the segment"
Yes
The current node = the end node of the segment;

Fig. 5. Block diagram representation of the second part of the improved recursive algorithm – recursive traversal and formulation of the operational condition

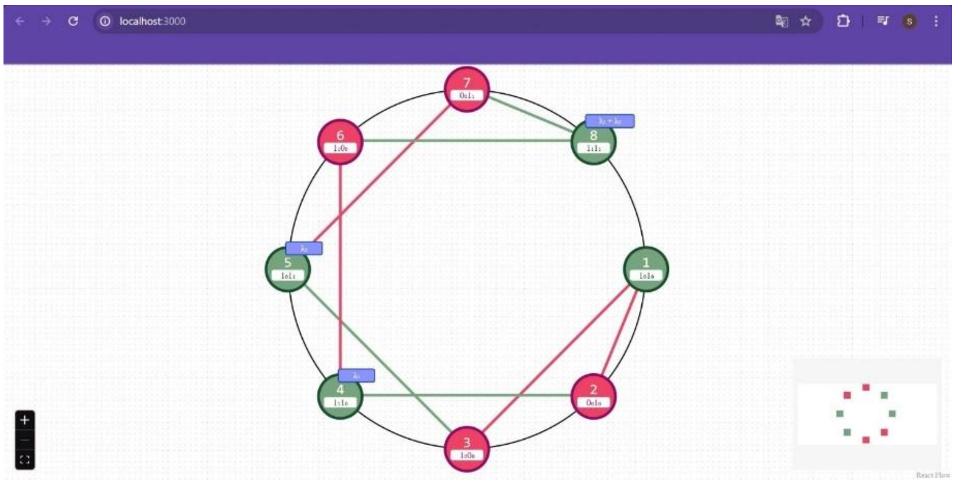

Fig 6. Implementation of the circular graph using React and ReactFlow

This page features a main area displaying the structural diagram of the system, with modules that have either a working state or a repair state (Fig. 7). There is also a menu on the page that allows navigation back to the main page or to the neighbouring states page. The neighbouring states page displays a graph containing three levels of elements: parent, current, and child (Fig. 8).

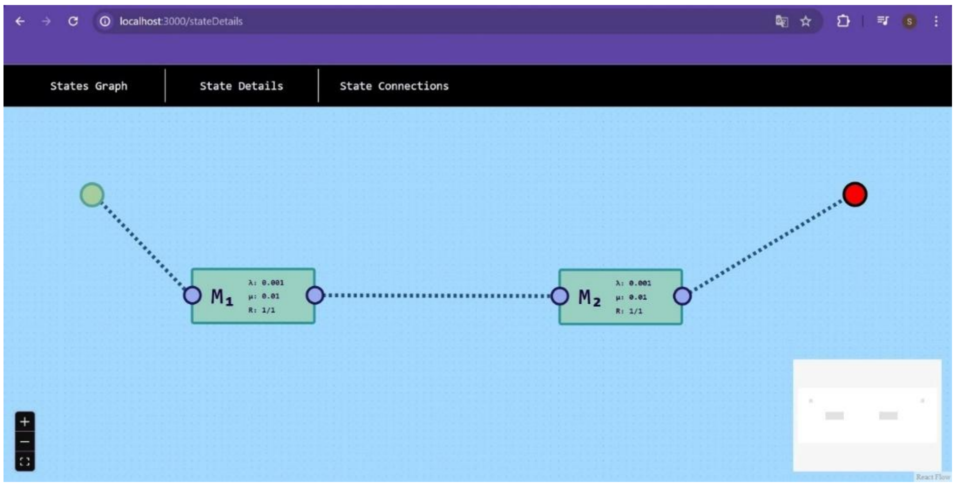

Fig. 7. Detailed information about the state

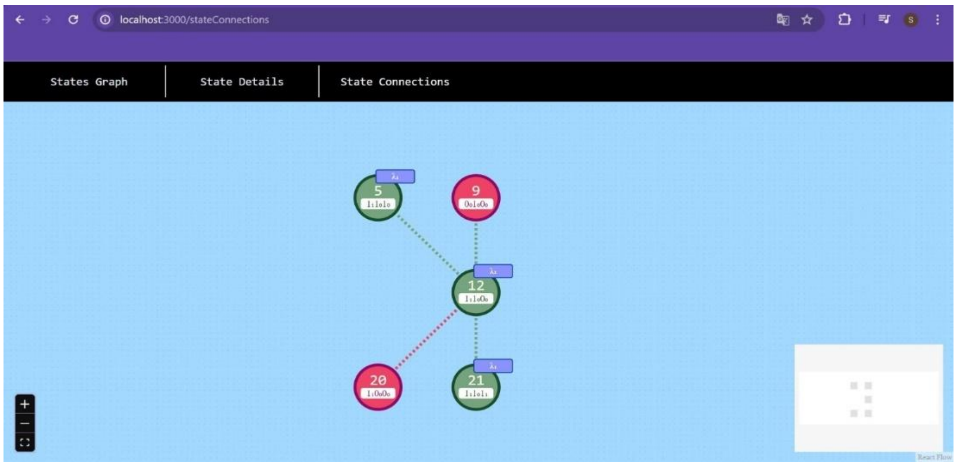

Fig. 8. Graph of neighbouring states

Animated arrows are designed to indicate the direction of state transitions. If the user double-clicks on an element of the neighbouring states graph, they will be able to navigate to the detailed information page for the selected state.

To verify the performance of this software, a series of experiments were conducted on reliability structural diagrams of various sizes and connection types (parallel and sequential). The performance of the improved recursive algorithm was compared with the method presented in [12].

Table 3

Comparison of performance for the reliability condition formulation: method [12] vs. improved recursive algorithm

Number of modules	Sequential connection		Parallel connection	
	Method [12], s	Improved recursive algorithm, s	Method [12], s	Improved recursive algorithm, s
1	0.0133530	0.0079854	0.0143543	0.0083983
2	0.0086102	0.0049453	0.0115355	0.0095868
4	0.0123576	0.0074434	0.0123048	0.0112578
8	0.0150145	0.0076454	0.0100280	0.0130272
10	0.0091833	0.0102026	0.0144573	0.0109297
15	0.0132885	0.0082445	0.0146141	0.0142705
20	0.0119692	0.0135530	0.0127616	0.0175789
30	0.0102165	0.0152295	0.0173087	0.0349637
50	0.0126795	0.0173358	0.0177975	0.1149013
75	0.0142591	0.0186773	0.0368207	0.4140173
100	0.0147048	0.0193757	0.0756552	0.8046696
150	0.0150907	0.0241673	0.1644606	2.7622245
200	0.0157585	0.0197763	0.3671252	6.1815201
350	0.0187226	0.0473787	1.8161816	34.7158770
500	0.0251735	0.0849076	5.4817717	99.6446662
600	0.0305666	0.1174315	9.2060068	168.5600439
620	0.0348336	0.1257890	10.7672005	188.7232709
640	0.0427135	0.1269443	11.7864979	219.2872035
650	–	–	–	–

According to the results of the experiments conducted, the improved recursive algorithm demonstrates better performance on small-scale diagrams (up to 10 elements for sequential connections and up to 15 elements for parallel connections). These results are graphically presented in Fig. 9.

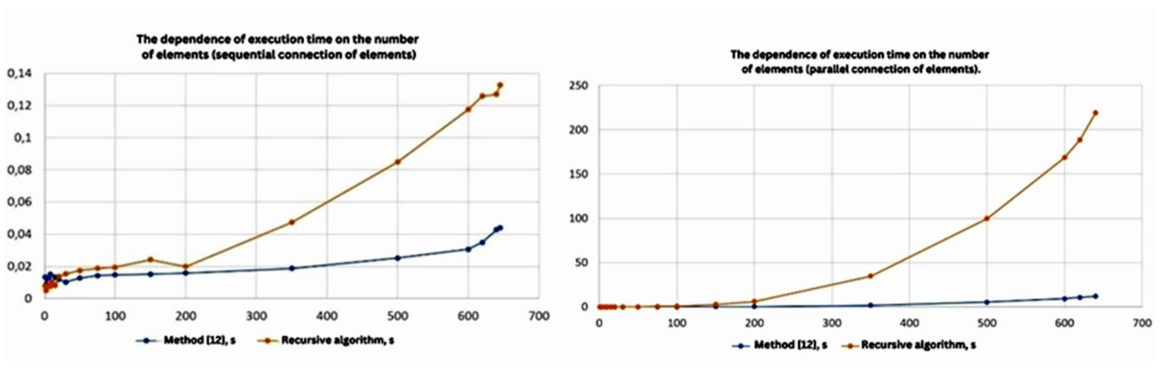

Fig. 9. Comparison of the performance for formulating the reliability condition between method [12] and the improved recursive algorithm

Additionally, testing of the performance speed for constructing circular and n-ary graphs using the developed software was conducted (Table 2).

Table 2

Comparison of performance speed in forming circular and n-ary state graphs

Number of modules	Sequential connection		Parallel connection	
	Circular graph, s.	N-ary graph, s.	Circular graph, s.	N-ary graph, s.
1	0.0078	0.0017314	0.0078	0.0017314
2	0.009	0.0016174	0.0189	0.0025893
3	0.014	0.004095	0.0423	0.0106329
4	0.024	0.0084683	0.1471	0.0503639
5	0.0512	0.0199266	0.5062	0.2615199
6	0.1067	0.0461575	1.8173	1.8756702
7	0.2205	0.1100145	7.136	11.3831539
8	0.451	0.3041155	–	–
9	0.9026	0.737526	–	–
10	2.7244	2.1262877	–	–
11	6.2412	5.9316393	–	–
12	18.1451	21.1073882	–	–

As evident from the experiments, the formation of circular and n-ary graphs have similar performance values, although with an increase in the number of elements, the formation of the n-ary graph is in some cases slower. Corresponding results are shown in Fig. 10.

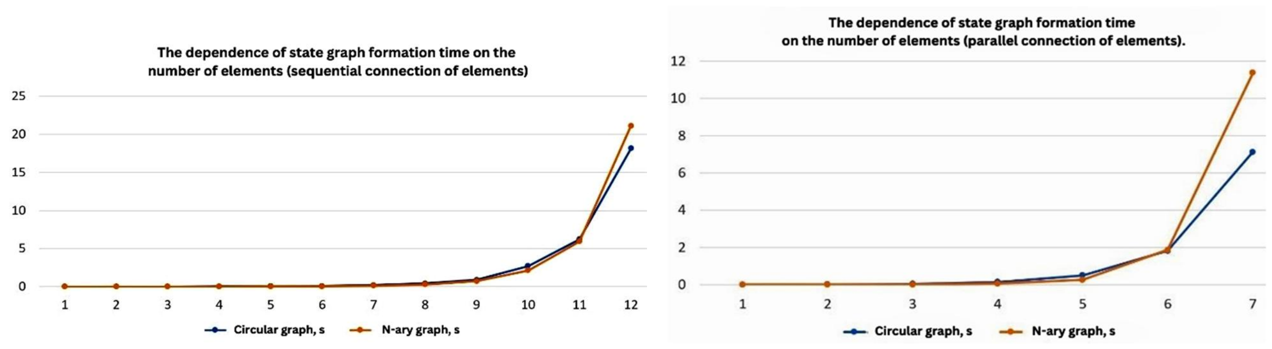

Fig. 10. Comparison of the performance of forming circular and n-ary state graphs

Conclusions

The improved software system for calculating the reliability indicators of complex technical systems has been implemented. The system was enhanced through the modification and software implementation of a recursive algorithm for formulating the operability condition and displaying a circular state/transition graph. Using the implemented system, it is possible to automate the construction of reliability block diagrams for technical systems, calculate the operability condition, and visualize this data using either n-ary or circular graphs.

Methods and tools for calculating the reliability indicators of complex technical systems, particularly software systems, have further developed, especially the algorithms and methods for automated formulation of the condition of operability. Additionally, tools for visualizing state/transition graphs for analysing reliability block diagrams have seen advancements. Furthermore, enhancements in the visualization tools for state/transition graphs have been implemented, allowing for improved analysis of reliability block diagrams.

The practical significance of the obtained results lies in the ability to use the improved recursive algorithm for the automated formulation of the condition of operability of reliability block diagrams. Its application on small-sized diagrams (up to 10 elements) yields a time gain of approximately 10 % in the case of sequential connections and about 32 % in the case of parallel connections.

The advanced software solution enables automated calculation of reliability indicators for software systems of any complexity level and reduces the influence of the human factor in the process of reliability design.

REFERENCES

1. TrustRadius. (n.d.). PTC Windchill overview. https://www.trustradius.com/products/ptc-windchill/reviews?qs=pros-and-cons#overview.

2. ALD Service. (n.d.). ALD reliability and safety software. https://aldservice.com/Download/download-reliability-and-safety-software.html.

3. ReliaSoft. (n.d.). BlockSim application introduction. https://help.reliasoft.com/blocksim21/content/current_application_intro.htm.

4. Isograph. (n.d.). Isograph AttackTree software. https://www.isograph.com/software/attacktree/.

5. Pérez-Rosés, H. (2018). Sixty years of network reliability. *Mathematics in Computer Science, 12*(3), 275–293. https://doi.org/10.1007/s11786-018-0345-5.

6. Song, K., Kim, Y. S., Pham, H., Chang, I. H. (2024). A software reliability model considering a scale parameter of the uncertainty and a new criterion. *Mathematics, 12*(11), 1641. https://doi.org/10.3390/math12111641.

7. Kumar, A. M. (2022). A Neuro-Fuzzy hybridized approach for software reliability prediction. *JUCS – Journal of Universal Computer Science, 28*(7), 708–732. https://doi.org/10.3897/jucs.80537.

8. Lin, J., Huang, C. (2022). Queueing-Based simulation for software reliability analysis. *IEEE Access, 10,* 107729–107747. https://doi.org/10.1109/ACCESS.2022.3213271.

9. Kim, Y. S., Pham, H., Chang, I. H. (2023). Deep-Learning Software Reliability Model using SRGM as activation function. *Applied Sciences, 13*(19), 10836. https://doi.org/10.3390/app131910836.

10. Haque, M. A., Ahmad, N. (2024). A logistic software reliability model with Loglog fault detection rate. *Iran Journal of Computer Science.* https://doi.org/10.1007/s42044-024-00192-x

11. Haque, M. A., Ahmad, N. (2022). A software reliability model using fault removal efficiency. *Journal of Reliability and Statistical Studies.* https://doi.org/10.13052/jrss0974-8024.1523.

12. Yakovyna, V. S., Seniv, M. M., Symets, I. I., Sambir, N. B. (2020). Algorithms and software suite for reliability assessment of complex teshnical systems. *Radio Electronics, Computer Science, Control, 4,* 163–177. https://doi.org/10.15588/1607-3274-2020-4-16.

УДОСКОНАЛЕНА ПРОГРАМНА СИСТЕМА РОЗРАХУНКУ ПОКАЗНИКІВ НАДІЙНОСТІ СКЛАДНИХ ТЕХНІЧНИХ СИСТЕМ

Максим Сенів[1], Степан Здебський[2]

Національний університет “Львівська політехніка”,
кафедра програмного забезпечення, Львів, Україна
[1] E-mail: maksym.m.seniv@lpnu.ua, ORCID: 0000-0003-1044-4628
[2] E-mail: stepan.zdebskyi.mnpzm.2024@lpnu.ua, ORCID: 0009-0000-4454-2539

Проаналізовано літературні джерела, в яких досліджено наявні методи та засоби розрахунку показників надійності складних технічних (зокрема, програмних) систем. Модель надійності сучасної комплексної технічної системи часто зображають у вигляді блок-схеми надійності

(reliability block diagram, RBD). Вона може містити декілька тисяч елементів, які почергово перебувають у різних станах (наприклад, робочий, відмовлений, відновлюваний). Це призводить до значного простору можливих станів у відповідній марковській моделі. Поведінку надійності системи прийнято описувати графом, вузли якого відповідають станам системи, а ребра – можливим переходам з одного стану в інший. Для автоматизації розрахунку показників надійності складних технічних систем використовується низка програмних продуктів. Проте цим продуктам притаманна низка недоліків, серед яких: складність впровадження у процеси проєктування і розроблення; значні витрати на придбання ліцензій та підготовку персоналу; відсутність сумісності з іншими продуктами аналізу надійності та управління життєвим циклом; відсутність інструментів для роботи із базами даних тощо. Більшість застарілих продуктів є desktop-додатками з недостатньо зручним графічним інтерфейсом. Основна мета цієї роботи – розроблення удосконаленої програмної системи, що передбачає модифікацію, імплементацію рекурсивного алгоритму формування умови працездатності та візуалізації кругового графа станів / переходів. За допомогою удосконаленої системи можна здійснювати автоматизовану побудову блок-схем надійності складних технічних, зокрема, програмних систем, обчислення умови працездатності за допомогою удосконаленого рекурсивного алгоритму та методу визначення умови працездатності, визначення станів системи і візуалізацію за допомогою парного або кругового графа. Також система надає інструменти для обчислення показників надійності: коефіцієнти готовності та простою, час напрацювання на відмову, параметри потоку відмов тощо. Удосконалений програмний комплекс дає можливість автоматизованого розрахунку показників надійності програмних систем довільного рівня складності та зменшує вплив людського фактору в процесі надійнісного проєктування.

Ключові слова: програмне забезпечення, блок-схема надійності, надійнісне проєктування, граф станів та переходів.