Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-83

OPTIMIZATION APPROACH TO SOLVING OF N-VERSION SOFTWARE SYSTEMS DESIGN PROBLEM

[1]Igor Kovalev, Dmitry Kovalev, Pavel Zelenkov, Anna Voroshilova, Natalia Ivleva]

Abstract **- The problem of developing an optimal structure of N-version software system presents a kind of very complex optimization problem. This causes the use of deterministic optimization methods inappropriate for solving the stated problem. In this view, exploiting heuristic strategies looks more rational. In the field of pseudo-Boolean optimization theory, the so called *method of varied probabilities* (MVP) has been developed to solve problems with a large dimensionality. Some additional modifications of MVP have been made to solve the problem of N-version systems design. Those algorithms take into account the discovered specific features of the objective function. The practical experiments have shown the advantage of using these algorithm modifications because of reducing a search space.**

Keywords **- N-version software, optimal structure, software system design, pseudo-Boolean optimization.**

I. INTRODUCTION

Development of high-reliable fault-tolerant systems is an interesting engineering problem having not only technical meaning but also social importance. Systems of this kind determine the stability in social and technical environments, and multiple examples of such systems' crashes prove the strong need for more reliable constructions which can be realized through the use of up-to-date methods and approaches.

The rapid progress of computer technique of late years has made the software an essential part of any complex automated system. The reliability of software component may determine the reliability of whole the hardware-software system. That's why during last years large attention is paid to the development of the methodologies of designing high-reliable software complexes [1-5].

Practically, multi-channel tools increasing the system reliability at the expense of a multiple duplication of certain structure elements are very much in evidence. This approach has given a good account of itself in the designing of hardware parts of complex systems. The use of this methodology leads to a sizable decreasing of appearance probability of random errors having the physical nature. In turn, this approach is not an influence on software reliability, since it doesn't trace so called dormant (or sleeping) errors which could arise while writing the program code by a stated specification [6].

The multi-version programming, as a methodology of the fault-tolerant software systems design, allows successful solving of the mentioned tasks. The idea of multi-version programming has been introduced by A. Avizienis in 1977 [7]. The term N-version programming (NVP) used in the literature is of equal meaning and often takes place in papers on the observed methodology. A. Avizienis introduced NVP as an independent generation of $N \geq 2$ functionally equivalent software modules from the same initial specification. The concurrent execution tools are provided for such the modules. In cross-check points (cc-points) software modules generate cross-check vectors (cc-vectors). The components of the cc-vectors and the cc-points are to be determined in the specification set.

The use of N-version programming approach turns out to be effective, since the system is constructed out of several parallel executed versions of some software module. Those versions are written to meet the same specification but by different programmers. Where, the writing process of each version of concrete software module in any way must not intersect with or depend on another version code writing. This is done to avoid the presence of same dormant (or sleeping) errors in separate software designs. This kind of errors is typical for software components.

The problem of developing the optimal structure of an N-version software system (NVS) is the following: to choose a set of software modules, so as to provide the highest reliability for the system subject to the budget constraint. Since a description of any possible system configuration is made through such the positioning of its components, we can say that an observed problem has the binary essence [8]. Moreover, the existing theory of pseudo-Boolean functions and their optimization contains strong tools for solving problems of this kind [9]. And that fact makes the use of binarization algorithms more affordable.

[1] Igor Kovalev, Dmitry Kovalev, Pavel Zelenkov, Anna Voroshilova, Natalia Ivleva

Institute of Computer Science and Telecommunications
Reshetnev Siberian State Aerospace University
Russian Federation

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-83

The process of a problem binarization consists in setting relationships between the system states and the binary space elements. In the case of our system model, we need to determine some Boolean vector the elements of which will characterize the system structure. Each element of such the Boolean vector will signify either presence or absence of corresponding system component [10].

In that way, before starting to describe the exact process of binarization, all the necessary terms should be coined and the presented system model should be overviewed in details.

II. OPTIMIZATION MODEL FOR STRUCTURING NVS

The structure of N-version software system is determined consisting of a set of tasks (a set $\mathbf{I}, \operatorname{card}\mathbf{I} = I$). All the tasks are divided into classes, i.e. a set of task classes is introduced as well ($\mathbf{J}, \operatorname{card}\mathbf{J} = J$).

To solve the tasks belonging to a certain class, there is a software module, which can be realized by any of its versions. Thus, $\mathbf{K}, \operatorname{card}\mathbf{K} = J$ – the set of software modules. Let us introduce the vector $\mathbf{S} = \{S_j\}\ (j = \overline{1, J})$, each component of which is equal to a number of module versions (S_j – the number of versions of module solving a task of class *j*) [11].

To describe the task belonging to particular classes, in [12] the authors define sets of tasks for every task class. That is introduced as two-dimensional array in programming terms. Since the numbers of tasks belonging to different classes are not equal, that may cause some difficulties when translating the analytic expressions into a program code.

Here, it is proposed to use only one set the capacity of which is equal to the number of tasks in a system. And each element of this set is equal to the number of class a task belongs to. So, the set $\mathbf{B}, \operatorname{card}\mathbf{B} = I$ is the set of class membership of tasks, i.e. the element B_i of the set **B** presents the number of class the *i*-th task belongs to.

Using the introduced notations, lets us determine a common analytic form of the number of versions solving *i*-th task. If the element B_i of the set **B** is the number of class the *i*-th task belongs to, then an element of the set **S**, the index number of which is equal to B_i, determines the number of versions in a module solving *i*-th task. Therefore, this number can be written like this S_{B_i}.

Basing on that, we will introduce the Boolean variables X_s^i to describe the control implication of different module versions:

$$X_s^i = \begin{cases} 1, & \text{the } s-\text{th } (s = \overline{1, S_{B_i}}) \text{ version of module } B_i \text{ is used} \\ & \text{to solve the } i-\text{th } (i = \overline{1, I}) \text{ task,} \\ 0, & \text{the } s-\text{th } (s = \overline{1, S_{B_i}}) \text{ version of module } B_i \text{ is } \mathbf{not} \\ & \text{used to solve the } i-\text{th } (i = \overline{1, I}) \text{ task.} \end{cases}$$

Expanding the introduced variables into the *implication vector* is the head moment in applying pseudo-Boolean optimization methods to the considered systems design.

Since a vector component number is specified by only one index and we deal with two-index variables, it is necessary to establish an algorithm forming an implication vector and an algorithm determining the component indices of this vector. Following section contains the algorithms to convert a problem of optimal structure design for N-version systems to a problem of pseudo-Boolean optimization and vice versa.

A. CONVERSION ALGORITHMS

The algorithm of an implication vector forming acts in the following way (see the scheme on fig. 2). The first component of an implication vector **X** describes the first version of a module to be involved in the solving of the first system task. If the software module which solves the first task has more than one version, the next component of vector **X** characterizes the second version of the first task module. In this way, all the versions of all software modules are overhauled.

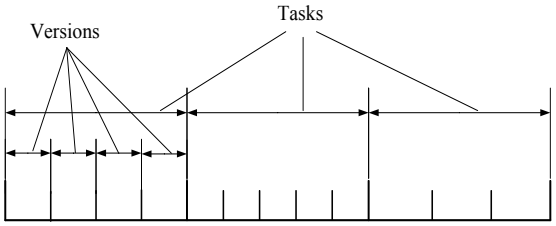

Figure 1. An example of the implication vector.

Hence, in order to determine the number of a vector **X** component, being aware of corresponding number of a task *i* and a number of version *s*, it is necessary to sum the number of versions in the modules solving the first (*i*-1) tasks and to add *s* to obtained value.

Analytically this conversion appears as follows:

$$pos = \begin{cases} s & , if\ i = 1, \\ \sum_{j=1}^{i-1} S_{B_j} + s & , if\ i > 1. \end{cases}$$

In order not to recalculate the first sum (the number of versions in the first (*i*-1) tasks) every time when optimizing a system, it would be better to count those sums depending on different *i* and to memorize them in an index vector:

$G_i = \sum_{j=1}^{i} S_{B_j}$, or in recurrent form

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-83

$$G_i = \begin{cases} S_{B_1} & , if\ i = 1, \\ G_{i-1} + S_{B_i} & , if\ i > 1. \end{cases}$$

Therefore, the value of the *i*-th component of vector **G** equals the number of versions in modules solving the tasks from the first up to the *i*-th. It results from this that the value of the last vector **G** component is equal to *n* – the implication vector dimensionality, i.e.

$$\sum_{i=1}^{N} S_{B_i} = n .$$

Once the index vector is introduced, the analytic record of a calculation of the implication vector component number takes the form of the following:

$$pos = \begin{cases} s & , if\ i = 1, \\ G_{i-1} + s & , if\ i > 1. \end{cases}$$

The reverse conversion task (a conversion of the implication vector component number to the numbers of task and version) consists of the consecutive determining of *i* and *s*. The flowchart of this algorithm is show on the fig. 2.

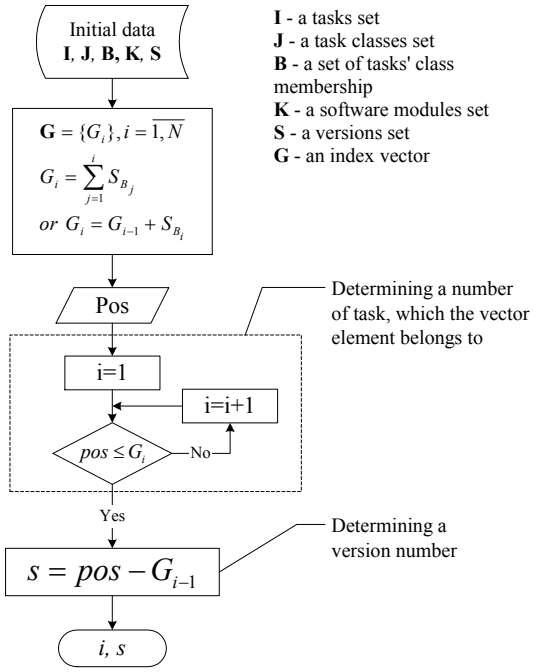

Figure 2. A conversion of the implication vector component number into the numbers of task and version.

Since the *i*-th element of the index vector equals the number of versions in modules solving the tasks from the first up to the *i*-th, the task number is determined by comparing the index of the implication vector component with the elements of the index vector. The comparison is being made from the first element of the index vector till the last one sequentially. And when the value of the parameter *pos* turns out to be less than or equal to the value of the index vector component, the required task number takes the value of this element number.

Then subtracting the number of versions in all the tasks from the first up to the (*i*-1)-th (equal to G_{i-1}) from *pos* we get a version number of software module solving the *i*-th task corresponding to *pos*.

The two presented algorithms are the core of applying binary approach to solve the stated problem. Thus, having received the tools for a problem conversion, it became possible to use the methods developed within the confines of pseudo-Boolean optimization study. Some the features of the considered problem are discussed in the following section. Basing on this the conclusions about the relevant methods are made.

B. THE MATHEMATICAL STATEMENT OF THE PROBLEM

The converting algorithms considered above allow to describe the NVS structure in a form of a Boolean vector. As it was noticed previously, the optimal design of control system is held subject to different parameters: the reliability (it should be as big as possible), the cost (it should be as small as possible or at least it shouldn't exceed some limit), the allocation & scheduling and so on [14].

In terms of optimization theory, a system reliability function of a system structure is nothing else but an objective function. And conditions imposed on the system structure are the constraints set to limit the objective function domain [13]. Since we are able to associate a system structure with a Boolean vector, an objective function is a pseudo-Boolean one. And an optimization problem becomes a pseudo-Boolean one too.

In the framework of the presented model we will use a system reliability function as the objective function and the system cost will be the constraint imposed on the system [15-19]. In analytic form this problem can be written as follows:

$$\max\ R = \prod_{i=1}^{I} R_i ,$$

where

$$R_i = 1 - \prod_{s=1}^{S_{B_i}} (1 - R_{B_i s})^{X_s^i}$$

subject to

$$\sum_{i=1}^{I} \sum_{s=1}^{S_{B_i}} X_s^i \cdot C_{B_i s} \le B .$$

Here, $R_{B_i s}$ and $C_{B_i s}$ are the reliability and the cost of the software version *s* from module which solves the task of class B_i

III. OPTIMIZATION ALGORITHMS TO FORM THE NVS STRUCTURE

To derive an optimal dependability solution by means of an systematic, the exhaustive comparison algorithm would mean that all potential system configurations have to be tentatively generated, checked for the fulfillment of the side conditions and processed to

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-83

compute the corresponding overall system reliability. This usually would cause a computing complexity that is untractable even for the most modern high speed computers: if, e.g., we consider a system consisting 64 modules, all of which are to be triplicated, thereby selecting each of the module versions from 5 different candidate modules, we would have to consider $[5!/3!\cdot 2!]^{64} = 10^{64}$ different system configurations! Assuming e.g. 1 nsec for processing each system configuration (of course, a value by far too optimistic!), the resulting 10^{55} sec of needed computation time would exceed the estimated age of the universe of about 10^{17} sec by many orders of magnitude! Therefore, here only stochastic search methods appear possible to provide, in a heuristic way, an optimal solution.

A. THE METHOD OF VARYING PROBABILITIES

In the field of pseudo-Boolean optimization theory, the so called *method of varied probabilities* has been developed to solve complicated problems, especially those ones with a large dimensionality [8]. The method of variable probabilities (MVP) presents a family of heuristic algorithms based on the common scheme: in order to find an extremal solution of a pseudo-Boolean optimization problem, a *probability vector* of dimensionality of sought solution vector is formed. Each component of the probability vector presents a probability of assigning a *one* value to the correspondent component of a Boolean vector. In the terms of developing NVS structure, it looks like a probability to include a version-candidate into the system structure.

The initial values of the probability vector components describe a situation when every software version has the equal probability to be included into the system structure. Then, at a computational phase, random decisions are generated according to the probability distribution specified by means of the probability vector. Each time the objective function is calculated in several random points, the values of the probability vector components are updated, so changing a probability distribution form. The way of changing these values defines a separate algorithms of MVP scheme. The common approach for updating a probability vector can be characterized by the rule: the better result received with a one-valued binary vector component the bigger probability is assigned for it to get the value of one in the final solution.

These scheme can be augmented whether by some special methods for updating the probability vector or through involving the peculiar procedures of generating random solutions at a computational phase of an algorithm. This paper discusses the two methods for updating the probability vector (ARSA and Modified ARSA ver. 1) and the two procedures of generating random solutions (the independent generation of random points and the generation of non-zero solutions) giving thus as a result four different realizations (algorithms) of MVP.

The Adaptive Random Search Algorithm (ARSA) plays a role of the background for the rest of the algorithms of MVP scheme [13]. Initially, ARSA has been developed for the problem of pattern recognition to select an informative subsystem of attributes. The main disadvantage of this algorithm is a potential problem of updating values of probability vector components. Namely, in some cases it is possible to get the values of intermediate solutions which do not let the probability vector components to be changed. To correct the defect, the modification of ARSA has been developed (Modified ARSA ver.1). The statistical data of applying the modified version of ARSA display the better behavior of the algorithm when solving problems of developing a structure of NVS.

Next, applying to the stated optimization problem, ARSA doesn't provide a technique of avoiding zero-solutions when solving the problem of designing NVS structure. To protect an algorithm against spending both computational and time resources for calculating the objective function values in the points of this kind, the particular technique of generating random non-zero solutions has been developed. This technique is utilized in the MVP based algorithm named NVS MVP (mentioning the strict field of using the algorithm).

Making use of both of the mentioned enhancements gave a great raise in the efficiency of applying the MVP based algorithms to the problem of NVS structure development. The statistical results presented in the final part of the paper show it. Different algorithms have been tasted on the same optimization problem with the same quantity of objective function calls.

The objective function of the presented optimization problem has several specific features which can assist to reduce a search domain, thus allowing to decrease the searching time. The objective function as a function of the whole system reliability represents the product of reliabilities of separate software modules. Consequently, when a reliability of any of the modules is equal to zero the overall system reliability turns into zero value also. Physically, it represents a case when there are no versions chosen for (at least) some of the software modules. The implication vector components corresponding to such the software modules will be assigned zeroes as well. Obviously, it is necessary to avoid computing the objective function in such the points.

The number of system structures having at least one software module without versions-candidates assigned can be determined as the difference of the number of all the possible structures and the quantity of the structures which provide every software module with at least one candidate, i.e. $N_0 = N_{all} - N_{R>0}$. The number of all possible structures is determined through the dimensionality of an implication vector *n* as follows $N_{all} = 2^n$. The second intermediate value is found basing on the multiplication principle from combinatorics as a number of all possible structures with software module combinations each without one of them (that with no versions assigned). Formally, it is described in the

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-83

following way: $N_{R>0} = \prod_{i=1}^{I}(2^{k_i} - 1)$, where I is the number of software modules, k_i represents a number of versions for the i-th software module.

Then, the final expression determining a sought value looks like this:

$$N_0 = 2^n - \prod_{i=1}^{I}(2^{k_i} - 1).$$

The value of this expression depends on an overall number of candidates (a dimensionality of the optimization problem), a number of software modules I and the numbers of versions for each of the software module (k_i, $i = \overline{1, I}$). In general case, this expression takes grand values counting up to $0.9N_{all}$, i.e. 90% of all the possible solutions. This means in this case that in order to find a solution it is sufficient to search through only 10% of the definitional domain of the objective function.

Unfortunately, there is the other side of the question making this result not so optimistic. Namely, for the problems of large dimensionalities reducing the search domain to 10% means diminishing the dimensionality of a problem by very small value. For instance, for a test problem of dimensionality n=117, avoiding all the null-valued points lowered the problem dimensionality only down to $n_{R>0} = 116$.

Nevertheless, exploiting this feature of the objective function has given satisfactory results when applying the algorithms of the method of varied probabilities (MVP). The modification of the MVP with the ability of avoiding null-valued points called NVS MVP has its own way of generating random points at iterational steps of the algorithms. In NVS MPV, random points are generated so that to provide each software module with at least one version.

At every iterational step, the whole implication vector generated is concerned as consisting of parts each describing the structure of a separate software module. Thus indeed, random vector generating consists of generating of random structures of modules. This approach allows having only non-zero solutions in result.

B. THE RANDOM SEARCH OF BOUNDARY POINTS

Another stochastic algorithm to optimize the structure of NVS is the algorithm of *random search of boundary points* [8]. It is based on the proved fact that a solution of the stated optimization problem is a so called *boundary point*. Or in terms of binary space topology, a point neighboring to the set of infeasible solutions. Such a point describes a system structure which can not be updated through including a software versions additionally without violating the resource conditions, i.e. no version can not be added to a system structure of this kind paying attention to restrictions. The algorithm of random search of boundary points constitutes a generating of multiple boundary solutions and comparing the objective functions values in them.

The constraint in this optimization problem partitions the whole binary space into two domains – the domain of solution satisfying the constraint function and a set of points not satisfying to the constraint. It is shown that these domains represent the connected sets and that a solution of correspondent optimization problem is a point neighboring to the set of infeasible solutions. This kind of solution can be called a *boundary point*.

Basing on the results stated above, it is clear that it is sufficient to search among only boundary points in order to find the best value of the objective function. Thus, the problem of finding a best solution becomes a problem of an exhaustive search on the boundary points set.

The following is the algorithm of generating boundary point for the problem of developing the optimal structure of NVS (Fig. 3).

Different boundary points can be reached using this algorithm when different combinations of ways to choose i at the second step of the algorithm will be followed.

Hence, the algorithm of searching boundary points will have the following scheme.

1. The initializing step: i=0.
2. Determine a boundary point $\mathbf{X}_{bi}$ (b – as an index means “boundary”).
3. Calculate the objective function value F_i=F($\mathbf{X}_{bi}$).
4. If the stopping condition is satisfied go to p. 5, otherwise i=i+1 and go to p. 2.
5. The solution is $F^* = \max_i F_i$.

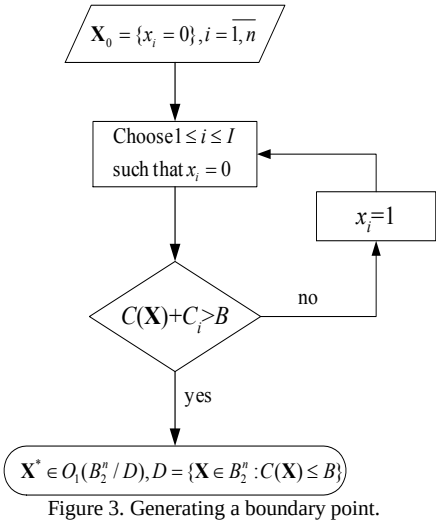

Figure 3. Generating a boundary point.

Separate variations of the *algorithm of boundary points search* may differ from each other in a stopping condition and in ways of reaching boundary when generating boundary points. For the optimization problems of high complexity it is more rational to use stochastic version of the algorithm when boundary points

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-83

are reached in a random way and this process is executed repeatedly.

IV. THE COMPARISON OF THE RANDOM OPTIMIZATION PROCEDURES

Concluding the paper, let us cite the comparative data of the computational results for different random optimization procedures. To gather such the information, the presented algorithms have been used to solve the test NVS structure optimization problems. The efficiency of the random search algorithms has been judged by the values of the objective and constraining functions.

The problem of dimensionality 117 has been chosen as the test problem, i.e. the developed software system included 117 software versions. It's worth mentioning that every of the random search algorithms needed approximately same period of time for calculating under equal conditions. That's why the time has not been set as an efficiency characteristic.

Table 1 contains the computational results of algorithms testing. The best searching capabilities have been revealed with the use of NVS MVP algorithm and the algorithm of boundary points search. The latter displayed the highest stability of the solutions found, although using NVS MVP it is sometimes possible to find more reliable system configurations.

№	Budget constraint B	Number of iterations	The random search algorithms			
			NVS MVP		Random search of boundary points	
			$R(X^*)$	$C(X^*)$	$R(X^*)$	$C(X^*)$
1.	800	15000	*0.7872*	789	*0.8074*	796
			0.7916	791	*0.7998*	797
			0.7907	791	*0.8177*	794
		30000	*0.8136*	786	*0.8318*	794
			0.8054	775	*0.8331*	797
			0.8118	784	*0.8377*	798
2.	900	15000	*0.9040*	850	*0.9149*	899
			0.9207	896	*0.9164*	899
			0.9039	887	*0.9148*	898
		30000	*0.9076*	867	*0.9192*	897
			0.9082	875	*0.9167*	897
			0.9155	890	*0.9177*	892
3.	1000	15000	*0.9701*	995	*0.9622*	993
			0.9523	986	*0.9609*	998
			0.9546	989	*0.9635*	998
		30000	*0.9651*	994	*0.9652*	997
			0.9554	988	*0.9661*	995
			0.9712	997	*0.9631*	996

Table 1. The results of random search algorithms working.

V. CONCLUSION

The problem of structuring an N-version software system is specified by the binary character, what made it plausible to apply the methods of pseudo-Boolean optimization. Within the limits of the discrete optimization a set of methods and algorithms has been proposed. The search capabilities of each of the algorithms realized have been investigated by solving the test problems. It was shown that the modification of the method of varying probabilities for NVS MVP together with the algorithm of boundary points search provide the best searching capabilities concerning the time efficiency and the solution quality.

References

[1] Laprie J.-C. et al. Hardware- and Software-fault tolerance: definition and analysis of architectural solutions// Proceedings of the IEEE, 1987, Pp. 116-121.

[2] Anderson T., Barrett P.A., Halliwell D.N., Moudling M.L., “An evaluation of software fault tolerance in a practical system”, Proc. Fault Tolerant Computing Symposium 1985, pp. 140-145.

[3] Scheer, S., Maier, T.: Towards Dependable Software Requirement Specifications. In: Daniel, P. (ed.) Proceedings of SAFECOMP 1997, New York (1997)

[4] Kovalev I. et al The control of developing a structure of a catastrophe-resistant system of information processing and control // In: 2015 IOP Conf. Ser.: Mater. Sci. Eng. 70 012008. doi:10.1088/1757-899X/70/1/012008.

[5] Kovalev I.V., Engel E.A., Tsarev R.Ju. Programmatic support of the analysis of cluster structures of failure-resistant information systems. Automatic Documentation and Mathematical Linguistics. 2007. T. 41. № 3. C. 89-92.

[6] Keene, S. J. Comparing Hardware and Software Reliability. Reliability Review, 14(4), December 1994, pp. 5-21.

[7] Avizienis A. The methdology of N-version programming// In: Software fault tolerance/ edited by M.R. Lyu, Wiley, 1995, Pp. 23-47.

[8] Antamoshkin A., Schwefel H.P., Torn A., Yin G. and Zilinskas A. System Analysis, Design and Optimization. Ofset Press, Krasnoyarsk, 1993.- 312 p.

[9] Antamoshkin, A.N. Random Search Algorithm for the p-Median Problem / A.N. Antamoshkin, L.A. Kazakovtsev // Informatica – 2013. - № 3(37). – P. 127–140.

[10] Lev Kazakovtsev, Predrag Stanimirovic, Idowu Osinga, Mikhail Gudima and Alexander Antamoshkin / Algorithms for location problems based on angular distances. Advances in Operations Research. – 2014. – Vol. 2014. Articale ID 701267. 12 pages. - http://www.hindawi.com/journals/aor/raa/701267/

[11] Kovalev I. Optimization problems when realizing the spacecrafts control// In: Advances in Modeling and Analysis, C, Vol. 52, No. 1-2, 1998, pp. 62-70.

[12] Kovalev I.V., Dgioeva N.N., Slobodin M.Ju. The mathematical system model for the problem of multi-version software design. Proceedings of Modelling and Simulation, MS'2004 AMSE International Conference on Modelling and Simulation, MS'2004. AMSE, French Research Council, CNRS, Rhone-Alpes Region, Hospitals of Lyon. Lyon-Villeurbanne, 2004.

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-83

[13] Kovalev I., Grosspietsch K.-E. Deriving the Optimal Structure of N-version Software under Resource Requirements and Cost/Timing Constraints// Proc. Euromicro' 2000, Maastricht, 2000, IEEE CS Press, pp. 200-207.

[14] Kovalev I. et al. The Minimization of Inter-Module Interface for the Achievement of Reliability of Multi-Version Software / Kovalev I., Zelenkov P., Ognerubov S // 2015 IOP Conf. Ser.: Mater. Sci. Eng. 70 012006. doi:10.1088/1757-899X/70/1/012006.

[15] Ashrafi N. et al. Optimal Design of Large Software-Systems Using N- Version Programming// IEEE Trans. on Reliability, Vol. 43, No. 2, 1994, pp. 344-350.

[16] Ashrafi N., Berman O. Optimization models for selection of programs, considering cost and reliability// IEEE Trans. on Reliability, Vol. 41, No. 2, 1992, pp. 281-287.

[17] Ashrafi N., Berman O. Optimization models for reliability of modular software systems// IEEE Trans. on Software Engineering, Vol. 19, No. 11, November 1993, pp. 1119-1123.

[18] Kovalev I. et al. Model of the reliability analysis of the distributed computer systems with architecture "client-server". IOP Conf. Series: Materials Science and Engineering 70 (2015) 012009. doi:10.1088/1757-899X/70/1/012009.

[19] Kovalev, I.V. Fault-tolerant software architecture creation model based on reliability evaluation / I.V. Kovalev, R.V.Younoussov; Advanced in Modeling & Analysis, vol. 48, № 3-4. Journal of AMSE Periodicals, 2002, pp. 31-43.