Technique of Selecting Multiversion Software System Structure with Minimum Simultaneous Module Version Usage

Denis V. Gruzenkin[1], Roman Yu. Tsarev[1], and Alexander N. Pupkov[1]

[1]Siberian Federal University, Krasnoyarsk, Russia
gruzenkin.denis@good-look.su, tsarev.sfu@mail.ru,
alex007p@yandex.ru

Abstract. Multiversion or N-version programming is well known as an effective approach, ensuring high level of software reliability. This approach is based on two fundamental strategies for enhancing the reliability of a software system – redundancy and diversity. Modules solving critical tasks are redundant and implemented in the form of functionally equivalent versions. In this connection versions can be developed by different programmer teams, in different languages, in different environment and can implement different methods and algorithms for solution of identical tasks in order to provide versions diversity. Complex software systems, as a rule, include a set of programs which can call the same modules for solving their target tasks, or to be more precise, versions of these modules. According to diversity concept call of different module versions allows to avoid identical failures. This article presents a technique of selecting optimal multiversion software system to minimize simultaneous usage of the same module versions.

Keywords: multiversion software, reliability, structure, interface

1 Introduction

At present, different methods of designing highly reliable software are known [1,2,3,4]. One of the most perspective approaches is multiversion or N-version programming (NVP), first presented by Avizienis and Chen [5]. It says that several programming components (versions) duplicating each other are included in the system. However, these versions are diverse, i.e. they implement different methods and algorithms to solve the same problem, they can be developed by different programmer teams, using different languages and different environment and so on [2], [6]. Multiversion implementation of program components provides system functioning regardless of hidden faults of some versions. The key advantage of N-version programming lies in the fact that system failure can occur only in case of considerable amount of versions failures [7]. Versions confirm each other's work which increases adequacy of results received [8].

adfa, p. 1, 2016.

N-version programming means that failures in functionally equivalent versions occur in different points, and thanks to this faults can be identified and resisted [9].

Use of module principle at a design stage is connected with the process of optimization of structure and correlations of independent multiversion software system components to achieve optimal parameters concerning development, debugging and exploitation of the system (see, for example, Kulyagin et al. [10]).

The set of tasks when selecting optimal structure of multiversion software systems includes the choice of optimal module set and data arrays, and system structure as a whole, formalized as a functional block scheme considering given technical and economical characteristics of system being developed.

Issues connected to optimization of the structure and components of multiversion software systems are contemplated in various works where different methods of this problem solution are presented. For example Kvasnica and Kvasnica in their work [11] use pseudoparallel optimization to form the structure of fault-tolerant software systems designed in line with the principle of the N-version programming approach. They propose several optimization procedures according to the features of the observed corresponding objective function.

Pham addresses the optimization issue for the cost of multiversion software systems and propose the solution for the minimum expected cost of multiversion software systems subject to the desired reliability level [12]. He also solves the problems of maximizing the reliability of the NVP subject to a constraint on expected system cost.

Redundancy can improve reliability, but increases the cost of software design and development. In [13] Rao et al. present a binary integer programming solution for multiversion software redundancy optimization.

Bhaskar and Kumar deal with the issues connected with criticality of the fault in multiversion program system and cost of its occurrence. Their work [14] suggests models for optimal release time under different constraints.

In [15] Kapur et al. propose a testing efficiency model incorporating the effect of imperfect fault debugging and error generation. Furthermore, they also formulate the optimal software release time problem for a 3-version software system under fuzzy environment and discuss a fuzzy optimization technique for solving the problem.

Probably, the greatest contribution to the solution of multiversion software optimization was made by Yamachi H., Yamamoto H. and Tsujimura Y. They formulate the problem of multiversion software system optimal design as a bi-objective 0-1 nonlinear integer programming problem optimization model, maximizing the system reliability and minimizing the system cost [16,17,18,19]. They solve the optimization problem using a multi-objective genetic algorithm employing the random-key representation to provide effective genetic search ability and the elitism and Pareto-insertion based on distance between Pareto solutions in the selection process [16,17,18,19].

In their works [20,21] Yamachi et al. formulate NVP design problem as the multi-objective optimization problem that seek Pareto solutions. In [20] they propose an algorithm that employs the breadth-first search method to find the Pareto solutions

under practical computation time. Further they proposed employing the branch-and-bound method to find the Pareto solutions [21].

Besides the above, we can note Levitin's works, where he uses genetic algorithms to form optimal structure of a fault-tolerant software systems built according to N-version programming principle [22,23].

Main criteria of synthesis of information processing module systems (multiversion software systems are undoubtedly such systems) alongside with reliability and cost at the stage of technical design can be: minimal intermodule interface complexity, minimum time exchange between operative and external computer memory when solving the task, minimal volume of unused data in exchange between operative and external computed memory, and maximum informational performance of the module system during solution of tasks. Values of these criteria, one way or another, depend upon structure of a multiversion software system, and particularly upon simultaneous use of versions of the same module by different multiversion software system programs.

This article considers the problem of maximization reliability with the limited cost of the system, along with minimization of simultaneous usage of the modules versions by different multiversion software system programs. It also offers a mathematical formalization of this problem. The problem solution is offered via recursive scheme of brute force (or exhaustive search), that allows to decrease the task solution time.

2 Problem's Statement

Let us consider the problem of selecting optimal multiversion software system structure which has high level of reliability, satisfies the given price constraints and provides minimal simultaneous usage of module versions.

This task appears at the stage of technical design which forms the common requirements to the software system, defines the system functions, procedures and data processing programs, including processing of input data, and getting intermediate and final results.

At module design, multiple versions which the module consists of are defined along with reliability of each version. On the basis of this data the module reliability can be calculated as follows [7,10]:

$$R_i(X_{ij}) = 1 - \prod_{j=1}^{m_i} (1 - R_{ij})^{X_{ij}},\ i = 1,\ldots,n, \tag{1}$$

where n – number of modules;

m_i – number of versions of module i;

R_{ij} – reliability of version j of module i;

X_{ij} – Boolean variable, equal to 1 if version j is used in module i, or 0 - otherwise. According to N-version programming principle:

$$\sum_{j=1}^{m_i} X_{ij} \geq 2, \; i = 1,\ldots,n.$$

Multiversion software module structure is shown in Fig. 1.

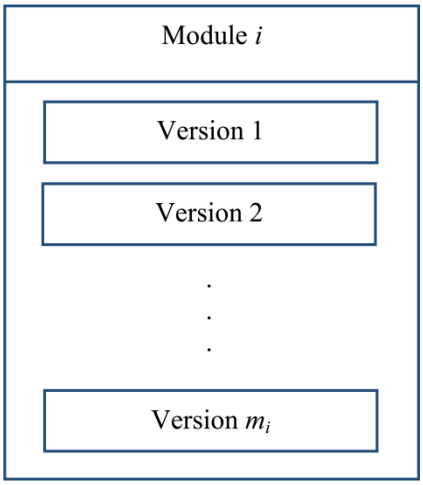

Fig. 1. Structure of multiversion module

Processes of control or data processing, for which multiversion software is developed, often contain complex mathematical calculations and process big data. Therefore, usually, complex software systems, instead of independent programs are needed. Figure 2 Illustrates multiversion software system structure.

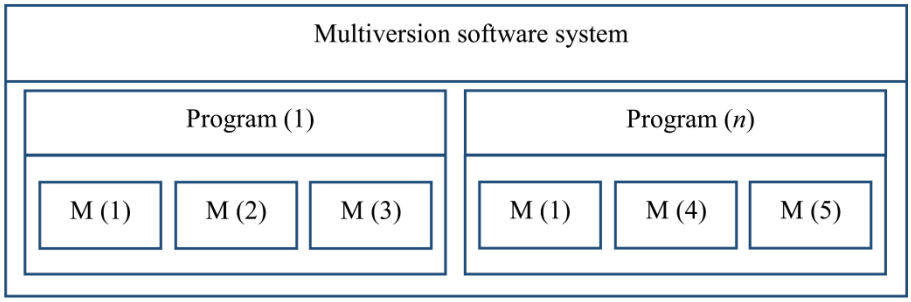

Fig. 2. Structure of multiversion software system

In Fig. 2, M (*i*) is module *i*. The problem of maximization of the multiversion software system reliability can be stated as follows:

$$\sum_{k=1}^{K} F_k \prod_{i \in S_k} R_i(X_{ij}) \to \max_{X_{ij}}, \; j = 1,\ldots,m_i, \tag{2}$$

where K – number of programs;

S_k – quantity of modules, corresponding to program k, $k = 1, ..., K$;

F_k – frequency of program k usage, $k = 1, ..., K$.

When designing multiversion software system structure let us solve the problem of minimum simultaneous usage of a separate module versions in different programs (Fig. 3) i.e. minimize the number of a module versions which are used in different programs.

For formalization of this task let us define additional variables:

W_{ik} – Boolean variable equals to 1 if module i is used by program k and 0 otherwise;

Y_{kj} – Boolean variable equals 1 if version j of the module is used by program k and 0 otherwise:

$$Y_{kj}(W_{ik}, X_{ij}) = \begin{cases} 1, \text{ if } \sum_{i=1}^{n} W_{ik} X_{ij} \geq 1, \\ 0, \text{ if } \sum_{i=1}^{n} W_{ik} X_{ij} = 0. \end{cases} \tag{3}$$

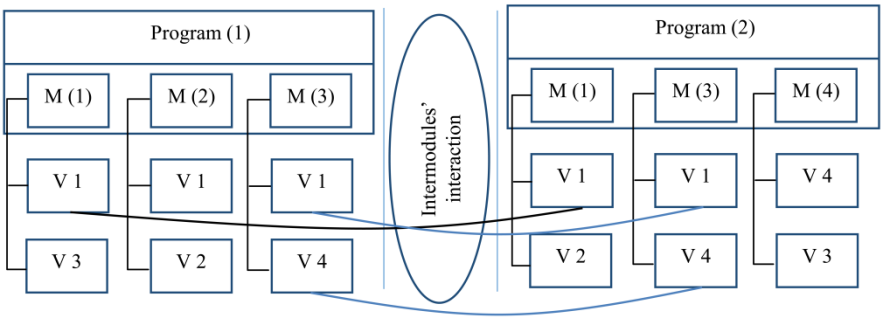

Fig. 3. The scheme of versions interaction in multiversion software system

Then the problem of selecting optimal multiversion software system structure with minimum simultaneous usage of the same modules by the multiverstion system programs can be formulated as follows:

$$\sum_{j=1}^{m_i} \sum_{k=1}^{K-1} \sum_{k'=k+1}^{K} Y_{kj}(W_{ik}, X_{ij}) \cdot Y_{k'j}(W_{ik'}, X_{ij}) \to \min_{W_{ik}, W_{ik'}, X_{ij}}, \; i = 1,\ldots,n. \tag{4}$$

This task can be solved using brute force by considering all variants of matrix X and recalculating matrix Y according to (1).

The NVP design problem is constructing a multiversion software system with maximum system reliability and within a given budget and known as an NP-complete problem [21].

Yamachi H. et al. claim in [21], that the NVP design problem can be formulated as a problem for maximizing reliability under the constraint of budget limitations:

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i} X_{ij} c_{ij} \le b,$$

where c_{ij} – cost of version j of module i, $j = 1, ..., m_i$, $i = 1, ..., n$;

b – budget available for system development.

Although, for such formulations, dynamic programming or genetic algorithms have been used, an algorithm that employs the branch-and-bound method can be applied as well [21].

In the example below the method of brute force has been used. This became possible due to limited number of programs, modules and versions of multiversion software system.

3 Algorithmic Basis for Method of Choosing a Multiversion Software System Structure

One of the approaches to defining an optimal structure of the designed multiversion software system is brute force method – exhaustive search of all possible variants of the modules versions usage in all programs of the system and selecting the best variant out of them.

The technique of choosing a multiversion software system structure includes the following steps.

On the basis of information about all available versions of the modules, the starting "basic" value of matrix X is set. Matrix X is created based on Boolean variables X_{ij}, equal to 1 in this case, if version j is available for use in module i, and zero if otherwise.

Brute force method allows to consider all possible values variants of designed multiversion software system structure. When doing so with all elements of Matrix X, the elements which starting value was equal to 1 change their value to zero and back, depending on the rate of duplicating the versions of a module in the structure of the multiversion software system given. Matrix X's elements which equal to zero originally, at brute force are not taken into consideration, and their values do not change.

At each stage of brute force method for a current variant of matrix X reliability of the modules (1) and multiversion software system as a whole are calculated (2).

Besides reliability analysis, each step of brute force method for each variant of matrix X involves calculation of matrix Y using the formula (3). After this, by using the criterion (4) we minimize simultaneous usage of the same versions of the modules by multiversion software system programs.

The problem of choosing a structure of multiversion software system can be considered as one of the following tasks:

- Maximization of reliability of a multiversion software system;
- Minimization of simultaneous usage of the modules version by multiversion software system programs;
- Maximization of reliability of multiversion software system with set level of modules versions usage by the programs;
- Minimization of simultaneous usage the modules versions with a set reliability of multiversion software system by the programs.

Naturally, solution of any of the abovementioned tasks corresponds to, a definite structure of multiversion software system which is formally introduced by means of matrix *X*.

However at brute force method a situation can arise when matrix *X* dimension will be big, and the amount of zeros in it will be drastically more than the amount of ones. In this case, calculation can take much time. For taking into account all possible combination of zeros and ones the recursion scheme is used, that is why a number of transitions grows exponentially with the increase of matrix *X*'s dimension.

To avoid unnecessary steps of the algorithm in recursion we suggest introducing some additional arrays:

1. Flat Boolean array *A*, dimension of which equals to a number of modules. This array takes into account the rate of version duplication. Element *i* of this array equals to 1, if the amount of versions, used by the module is more than the rate of duplication, otherwise element *i* equals to zero. This allows to ignore the rows of matrix *X* where the original number of versions in the module equals to the rate of duplication.

2. Two-dimensional array *B* with the dimension of matrix *X*. Elements of this array are transition structures organized as follows: {number of row of matrix *X*, number of column of matrix *X*}. Each element of this array contains indexes of a row and a column of the following "basic element" (i.e. one), i.e. points to the index of the element to be transferred to at the next step. If *X* [*i, j*] – the last "basic element", so the corresponding indexes in the structure of the given array will be equal to -1. If during recursion a transition to element *X* [-1, -1] is encountered, this means that the final variant of matrix *X* has been obtained and we can move to calculations of the system parameters.

Array *B* is set as follows: we shall define indexes of a row and column in matrix *X* for each element *B* [*i*, *j*] by moving from element to element along columns and rows until we find a "basic element". If the encountered "basic element" is in a row with index *i* and value of element *i* of array *A* equals to zero, then we move to the next row. If at reaching the end of the array the "basic element" has not been found, then the elements of array *B* which correspond to transition structures are set to -1.

The suggested approach efficiency is illustrated in Fig. 4 and 5. Figure 5 presents an example of transitions along matrix *X* with the rate of duplicating equal to 2.

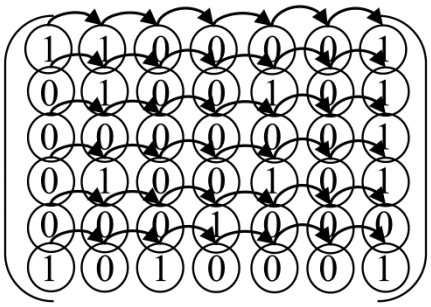

Fig. 4. Sequence of transitions at brute force method

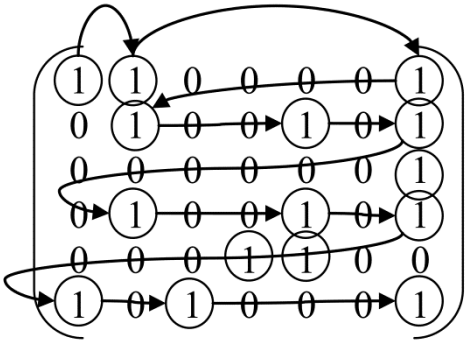

Fig. 5. Transitions by elements with the rate of duplicating equal to 2

Thus the suggested above approach lessen the number of recursive transition and decrease time needed for analysis of all possible variants of the multiversion software system structure.

4 Results and Discussion

Let us consider an example of selection an optimal structure for multiversion software system. The general structure of multiversion software system is given in Fig. 6. Here all versions available for each module are presented, the solid lines define the modules used in different programs of multiversion software system. In table 1 values of reliability indices of the modules versions and frequency of usage by the programs are listed.

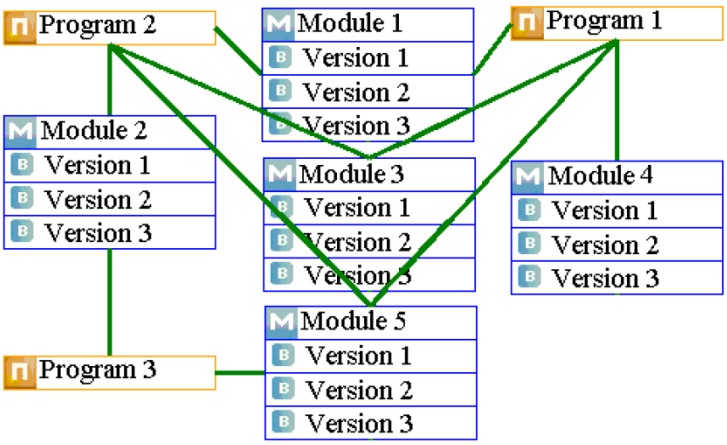

Fig. 6. Redundant structure of multiversion software system

Table 1. Input Data

Program	Module	Version	Version Reliability	Frequency of Program Usage
P (1)	M (1)	V 1.1	0.950	0.6
		V 1.2	0.920	
		V 1.3	0.925	
	M (3)	V 3.1	0.960	
		V 3.2	0.860	
		V 3.3	0.930	
	M (4)	V 4.1	0.930	
		V 4.2	0.950	
		V 4.3	0.900	
	M (5)	V 5.1	0.960	
		V 5.2	0.910	
		V 5.3	0.950	
P (2)	M (1)	V 1.1	0.950	0.69
		V 1.2	0.920	
		V 1.3	0.925	
	M (2)	V 2.1	0.900	
		V 2.2	0.930	
		V 2.3	0.950	
	M (3)	V 3.1	0.960	
		V 3.2	0.860	
		V 3.3	0.930	
	M (5)	V 5.1	0.960	
		V 5.2	0.910	
		V 5.3	0.950	
P (3)	M (2)	V 2.1	0.900	0.5
		V 2.2	0.930	
		V 2.3	0.950	
	M (5)	V 5.1	0.960	
		V 5.2	0.910	
		V 5.3	0.950	

On the grounds of input data two tasks have been solved: maximization of multiversion software system reliability and minimization of simultaneous use of the same modules versions by multiversion software system programs.

Implementation of the above technique gave the following results: when solving the first task reliability of multiversion software system being designed constituted 0.99897; at the same time a number of simultaneous usage of the same versions of the modules in different programs is equal to 18.

When solving the second task reliability of multiversion software system being designed was 0.98837, while only six versions of the modules were used in different multiversion software system programs.

It is obvious that the given values of reliability and simultaneous usage of the versions by multiversion software system programs are threshold values for this example. With any other problem statement these values can not be excelled.

It should be noted that the size of the task allowed to implement the suggested technique based on a recursive scheme of brute force method. In case of big size of the task genetic algorithms or dynamic programming can be applied.

Moreover, we can consider the task of selecting an optimal structure of multiversion software system as a bi-objective one. This way the important things turn out to be the problem of balance between reliability and minimal level of the module versions usage by different multiversion software system programs. In this work such task was not considered, however solution of this task can become a subject for further research.

5 Conclusion

N-version programming allows reaching a maximum reliability of a software system. However, designing a redundant software system is not a trivial problem, demanding solving the set of tasks and considering different constraints.

This article represents the problem of choosing multiversion software system structure, which is defined with the usage of different versions of the same modules by system programs, is represented. The need for solving this problem is conditioned with the diversity principle realization to avoid potentially identical faults in different module versions.

The article also provides a technique allowing to solve the problem of selecting an optimal multiversion software system structure successfully considering requirements to its reliability, cost and structure constraints, structure complexity and simultaneous usage of module versions by multiversion system programs.

To solve the task of choosing multiversion software system structure involving minimal simultaneous use of the module versions the authors offered a modification of recursive scheme of exhaustive search, allowing to decrease the number of steps during search of the variants considered.

References

1. Carzaniga, A., Mattavelli, A., Pezze, M.: Measuring Software Redundancy. In: 37th IEEE International Conference on Software Engineering (ICSE), pp.156-166, IEEE/ACM (2015)
2. Popov, P., Stankovic, V., Strigini, L.: An Empirical Study of the Effectiveness of "Forcing" Diversity Based on a Large Population of Diverse Programs. In 23rd IEEE International Symposium on Software Reliability Engineering (ISSRE), pp.41-50 (2012)
3. Salewski, F., Kowalewski, S.: Achieving highly reliable embedded software: An empirical evaluation of different approaches. In 26th International Conference on Computer Safety, Reliability, and Security, SAFECOMP, pp. 270-275, Nuremberg (2007)
4. Son, H.S., Koo, S.R.: Software Reliability Improvement Techniques. Springer Series in Reliability Engineering, 23, 105-120 (2009)
5. Avizienis, A., Chen, L.: On the implementation of N-version programming for software fault-tolerance during program execution. In Proc. IEEE Comput Soc Int Comput Software & Appl Conf, COMPSAC, pp. 149-155 (1977).
6. Durmuş, M.S., Eriş, O., Yildirim, U., Söylemez, M.T.: A new bitwise voting strategy for safety-critical systems with binary decisions. Turkish Journal of Electrical Engineering and Computer Sciences, 23 (5), pp. 1507-1521 (2015)
7. Kapur, P.K., Pham, H., Gupta, A., Jha, P.C.: Software Reliability Assessment with OR Applications. Springer-Verlag London Limited (2011)
8. Latif-Shabgahi, G., Bass, J.M., Bennett, S.: A taxonomy for software voting algorithms used in safety-critical systems. IEEE Transactions on Reliability, 53, 319–328. (2004)
9. Sommerville, I.: Software engineering; Pearson; 9 edition, Addison-Wesley (2011)
10. Kulyagin, V.A., Tsarev, R.Yu., Prokopenko, A.V., Nikiforov, A.Yu., Kovalev, I.V.: N-version design of fault-tolerant control software for communications satellite system. In International Siberian Conference on Control and Communications (SIBCON), pp.1-5 (2015)
11. Kvasnica, P., Kvasnica, I.: Parallel modelling of fault-tolerant software systems. International Review on Computers and Software, 7 (2), pp. 621-625. (2012)
12. Pham, H.: On the optimal design of N-version software systems subject to constraints. The Journal of Systems and Software, 27 (1), pp. 55-61. (1994)
13. Rao, N.M., Goura, V.M.K.P., Roy, D.S., Mohanta, D.K.: A binary integer programming solution for optimal reliability of computer relaying software incorporating redundancy. In Proc. IEEE Recent Advances in Intelligent Computational Systems, RAICS, pp. 524-527 (2011)
14. Bhaskar, T., Kumar, U.D.: A cost model for N-version programming with imperfect debugging. Journal of the Operational Research Society, 57 (8), pp. 986-994. (2006)
15. Kapur, P.K., Gupta, A., Jha, P.C.: Reliability growth modeling and optimal release policy under fuzzy environment of an N-version programming system incorporating the effect of fault removal efficiency. International Journal of Automation and Computing, 4 (4), pp. 369-379. (2007)
16. Yamachi, H., Tsujimura, Y., Yamamoto, H.: Pareto distance-based MOGA for solving Bi-objective N-version program design problem. Advances in Soft Computing, (AISC), pp. 412-422. (2005)
17. Yamachi, H., Yamamoto, H., Tsujimura, Y.: Multiobjective evolutionary optimal design of N-version software system. Advances in Safety and Reliability – Proc. of the European Safety and Reliability Conference, ESREL, 2, pp. 2053-2060. (2005)

18. Yamachi, H., Tsujimura, Y., Yamamoto, H.: Evaluating the effectiveness of applying genetic algorithms for NVP system design. Journal of Japan Industrial Management Association, 57 (2), pp. 112-119. (2006)
19. Yamachi, H., Tsujimura, Y., Kambayashi, Y., Yamamoto, H.: Multi-objective genetic algorithm for solving N-version program design problem. Reliability Engineering and System Safety, 91 (9), pp. 1083-1094. (2006)
20. Yamachi, H., Yamamoto, H., Tsujimura, Y., Kambayashi, Y.: Searching Pareto solutions of bi-objective NVP system design problem with breadth first search method. In Proc. 5th IEEE/ACIS Int. Conf. on Comput. and Info. Sci., ICIS, pp. 252-258. (2006)
21. Yamachi, H., Yamamoto, H., Tsujimura, Y., Kambayashi, Y.: An algorithm employing the branch-and-bound method to search for Pareto solutions of Bi-objective NVP system design problems. Journal of Japan Industrial Management Association, 58 (1), pp. 44-53. (2007)
22. Levitin, G.: Optimal structure of fault-tolerant software systems. Reliability Engineering and System Safety, 89 (3), pp. 286-295. (2005)
23. Levitin, G., Ben-Haim, H.: Genetic algorithm in optimization of fault-tolerant software. Advances in Safety and Reliability - Proceedings of the European Safety and Reliability Conference, ESREL, 2, pp. 1259-1265. (2005)