

Terskov V.A., Sheenok D.A., Kartsan I.N.

GENETIC ALGORITHM FOR OPTIMIZATION OF ONBOARD SOFTWARE ARCHITECTURE

The paper describes the model of onboard software architecture that allows us to assess the dependability of a designed program system and labour efforts spent on its realization. The paper defines parameters and operators of the genetic algorithm that makes it possible to find optimal characteristics of onboard software architecture.

Keywords: *genetic algorithm, onboard software, program redundancy, software dependability.*

Onboard software (OSW) in automated control systems to control satellite communication systems is implemented on various types of computers whose characteristics are defined by the purpose of systems [1]. These factors influence a rational level of designing automation, labor input and duration of software development, etc. However, principles and methods of SW designing vary relatively little.

OSW used in automatic control systems to control satellite communication systems possesses all properties of complex systems [2]. It contains a large quantity of modules closely interacting during common target task solution.

OSW has the common objective of functioning – information processing and decision-making to control objects, right up to generation of corresponding control actions [3].

Hierarchical structures with several levels of grouping and subordination of modules are widely used for ensuring interaction of modules in an integrated complex. Each module has the target task and specific individual criterion of quality, as a rule, not corresponding to the efficiency criterion of the whole system of programs.

It is believed that choosing this or that alternative of construction of onboard software architecture (OSW), one should be guided by dependability criteria and costs for realization of a system with specified dependability. It is obvious that statements of these criteria disagree with each other as a system with a greater dependability demands larger resources.

The dependability index of a system in the initial model of program system architecture is the value of system availability predicted at the design stage. The dependability of the whole system can be achieved by introducing program redundancy in separate components of its architecture. The basic resource in OSW development is labor expenditures of experts implementing the system. To calculate these indices in a model, it is necessary to take into account additional parameters.

In a component of program architecture, whose functioning is especially critical for dependability, program redundancy can be introduced by the method of N-version programming or by recovery block. It is obvious that dependability of components with program redundancy is directly proportional to the depth of redundancy (or to the number of its various versions) [4] and dependability of the environment of versions' execution (algorithm of voting or acceptance test).

Dependability of multiversion component i at the architectural level j constructed of K versions by the method of multiversion programming for any K is equal to [5]:

$$R_{ij} = p_{ij}^{v}\left(1-\prod_{k=1}^{K}(1-p_{ij}^{k})\right),$$

where p_{ij}^{v} is the probability of non-failure operation of a voting algorithm, p_{ij}^{k} is the probability of non-failure operation of the version $k \in Z_{ij}$.

Dependability of the multiversion component i at the architectural level j constructed of K versions by recovery block method for any K is equal to [5]:

$$R_{ij} = \sum_{k=1}^{K}\left(p_{ij}^{k} p_{ij}^{AT}\right)\prod_{l=1}^{k-1}\left((1-p_{ij}^{l})p_{ij}^{AT} + p_{ij}^{l}(1-p_{ij}^{AT})\right),$$

where p_{ij}^{AT} is the probability of non-failure operation of the acceptance test for a component i, i=1.., N at the level j, j=1.., M; p_{ij}^{k} is the probability of non-failure operation of the version $k \in Z_{ij}$.

The probability of failure of the component i at the level j is equal to:

$$PF_{ij} = 1 - R_{ij}.$$

Labor input of system engineering is equal to the sum of labor inputs of all components of system architecture. During calculation of labor input of components' development with program redundancy, the costs for implementation of versions' execution environment of the module (NVX_{ij}) and the costs for implementation of each version of the i-th component at the level j (T_{ij}^{k}) [6] should be taken into account. Labor input of the whole system development of T_s is calculated as follows:

$$T_S = \sum_{j=1}^{M}\sum_{i=1}^{N_j}\left(B_{ij}T_{ij} + \left(NVP_{ij} + RB_{ij}\right)\left(NVX_{ij} + \sum_{k\in Z_{ij}}^{K} T_{ij}^{k}\right)\right).$$

where NVX_{ij} is the labor input of versions execution environment development (the acceptance test for *RB* or voting algorithm for *NVP*) (*N-version execute environment*); B_{ij} is the binary variable accepting the value 1 (then NVP_{ij}=0, RB_{ij}=0) if the program component does not use program redundancy, otherwise it is equal to 0;

NVP_{ij} is the binary variable accepting the value 1 (then B_{ij}=0, RB_{ij}=0) if program redundancy is introduced in the program component by *NVP* method, otherwise it is equal to 0;

RB_{ij} is the binary variable accepting the value 1 (then B_{ij}=0, NVP_{ij}=0) if program redundancy is introduced in the program component by *RB* method, otherwise it is equal to 0.

In case if we model an already existing system, which is planned to be updated, then for those architectural components which already exist in system, costs are equal to 0 (T_{ij}=0).

The program component participating in critical control cycles should execute calculations in such time that the time of the whole control cycle should not exceed the critical value. If the component is not in time to give out control action to the other component, then a failure occurs.

The average time of the execution of the program component i at the level j is calculated as the sum of the component operating time without failures and the average idle time of the component:

$$RT_{ij} = TU_{ij} \cdot Ntu_{ij} \cdot \left(1 - \left(PF_{ij} + \sum_{ab \in E_{ij}} \left(PL_{ij}^{ab} \cdot PF_{ab} \cdot PU_{ab}\right)\right)\right) +$$

$$+ \left(TA_{ij} \cdot Nta_{ij} + TC_{ij} \cdot Ntc_{ij} + TE_{ij} \cdot Nte_{ij}\right) \cdot \left(PF_{ij} + \sum_{ab \in E_{ij}} \left(PL_{ij}^{ab} \cdot PF_{ab} \cdot PU_{ab}\right)\right),$$

where PL_{ij}^{ab} is the conditional probability of a failure occurring in the component i at the level j if a failure occurs in the component a at the level b, $a \in \{1,..,N_b\}$, $b \in \{1,..,M\}$, $i \in \{1,..,N_j\}$, $j \in \{1,..,M\}$;

PU_{ab} is the probability that the component a at the level b will be used;

PF_{ab} is the probability of a failure occurring in the component a at the level b;

Nta_{ij} is the number of faulty components at lower architectural levels during access to the component i at the level j;

Ntc_{ij} is the number of faulty components at all architectural levels analyzed at the same time with the component i at the level j;

Nte_{ij} is the number of faulty components at all levels of architecture in which there is an elimination of failures during the elimination of a failure in the component i at the level j;

Ntu_{ij} is the number of components at all levels of architecture used at the same time with the component i at the level j.

Below the designations of parameters of OSW architecture model [7] are presented:

M – the number of architectural levels in program architecture;

N_j – the number of components at the level j, $j \in \{1,\ldots M\}$;

D_{ij} – a set of indexes of components dependent on the component i at the level j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$;

E_{ij} – a set of indexes of components on which the component i at the level j depends, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$;

F_{ij} – a failure event occurred in the component i at the level j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$;

PU_{ij} – the probability that the component i at the level j will be used;

PF_{ij} – the probability of a failure occurring in the component i at the level j;

R_{ij} – the probability of a failure not occurring in the component i at the level j;

PL_{nm}^{ij} – the conditional probability that a failure will occur in the component m at the level n if there is a failure in the component i at the level j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$, $n \in \{1,\ldots N_m\}$, $m \in \{1,\ldots M\}$;

TA_{ij} – relative access time to the component i at the level j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$, determined by the ratio of the average access time to the component i at the level j to the number of faulty components at lower levels of architecture during access to the component; parameter TA_{ij} can be used, if the component operates on the remote or isolated systems;

Nta_{ij} – the number of faulty components at lower levels of architecture during access to the component i at the level j;

TC_{ij} – the relative time of failure analysis in the component i at the level j, determined by the ratio of the failure analysis average time in the component i at the level j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$, to the number

of faulty components at all levels of architecture analyzed simultaneously (time for error reproduction and its localization relates to the analysis time);

Nta_{ij} – the number of faulty components at all levels of architecture analyzed simultaneously with the component i at the level j;

TE_{ij} – the relative time of failure removal in the component i at the level j, determined by the ratio of restoration average time in the component i at the level j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$, to the number of faulty components at all levels of architecture in which removal of failures occurs at the same time;

Nte_{ij} – the number of faulty components at all architecture levels in which removal of failures occurs during removal of failure in the component i at the level j;

TU_{ij} – the relative time of use of the component i at the level j, determined by the ratio of average use time of the component i at the level j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$, to the number of components at all levels of architecture used simultaneously;

Ntu_{ij} – the number of components at all levels of architecture used simultaneously with the component i at the level j;

K_{ij} – the program redundancy depth of the component i at the level j;

Z_{ij} – the plurality of versions of the component i at the level j;

RT_{ij} – the average time of execution of the module i at the level j;

T_{ij} – the labor input of the development component i at the level j;

T^k_{ij} – the labor input of the development version k of the component i at the level j, $k \in Z_{ij}$, in man-hours;

NVX_{ij} – the labor input of development of versions' execution environment (the acceptance test for *RB* (*recovery block*) or voting algorithm for *NVP* (*N-version programming*));

B_{ij} – the binary variable accepting value 1 (in this case $NVP_{ij}=0$, $RB_{ij}=0$) if the program component does not use program redundancy, otherwise it is equal to 0;

NVP_{ij} – the binary variable accepting value 1 (in this case $B_{ij}=0$, $RBij=0$) if program redundancy is introduced in the program component by *NVP* method, otherwise it is equal to 0;

RBij – the binary variable accepting value 1 (in this case $B_{ij}=0$, $NVP_{ij}=0$) if program redundancy is introduced in the program component by *RB* method, otherwise it is equal to 0;

TR – the average idle time of a system which is defined as the average time during which the software cannot carry out its functions;

MTTF – the mean time to failure in system which is defined as the average time during which failures in the software do not arise;

S – the availability factor of a program system;

T_s – the general labor input of program system realization.

Mean time to failure in a program system is determined as follows [8]:

$$MTTF = \sum_{j=1}^{M} \sum_{i=1}^{N_j} \{PU_{ij} \times (1 - PF_{ij}) \times [TU_{ij} +$$

$$\sum_{(m=1)\&(m\neq j)}^{M} \sum_{n=1}^{N_m} \left((1 - PL_{nm}^{ij}) \times \left(TU_{nm} + \sum_{l \in D_{nm}} \left((1 - PL_{lm}^{nm}) \times TU_{lm} \right) \right) \right) +$$

$$+ \sum_{k \in D_{ij}} \left((1 - PL_{kj}^{ij}) \times \left(TU_{kj} + \sum_{(m=1)\&(m\neq j)}^{M} \sum_{n=1}^{N_m} \left((1 - PL_{nm}^{kj}) \times \left(TU_{nm} + \sum_{l \in D_{nm}} \left((1 - PL_{lm}^{nm}) \times TU_{lm} \right) \right) \right) \right) \right)] \}$$

Average idle time of a program system is equal to [8]:

$$TR = \sum_{j=1}^{M} \sum_{i=1}^{Mj} \{PU_{ij} \times PF_{ij} \times [(TA_{ij} + TC_{ij} + TE_{ij}) +$$

$$+ \sum_{(m=1)\&(m\neq j)}^{M} \sum_{n=1}^{N_m} \left(PL_{nm}^{ij} \times \left((TA_{nm} + TC_{nm} + TE_{nm}) + \sum_{l \in D_{nm}} \left(PL_{lm}^{nm} \times (TA_{lm} + TC_{lm} + TE_{lm}) \right) \right) \right) \times$$

$$\times \sum_{k \in D_{ij}} [PL_{kj}^{ij} \times [(TA_{kj} + TC_{kj} + TE_{kj}) +$$

$$+ \sum_{(m=1)\&(m\neq j)}^{M} \sum_{n=1}^{N_m} \left(PL_{nm}^{kj} \left((TA_{nm} + TC_{nm} + TE_{nm}) + \sum_{l \in D_{nm}} \left(PL_{lm}^{nm} \times (TA_{lm} + TC_{lm} + TE_{lm}) \right) \right) \right)]]]\}.$$

The availability factor readiness of a program system is determined as follows:

$$S = \frac{MTTF}{MTTF + TR}.$$

Man-hours for system engineering are equal to:

$$T_S = \sum_{j=1}^{M} \sum_{i=1}^{N_j} \left(B_{ij} T_{ij} + \left(NVP_{ij} + RB_{ij} \right) \cdot \left(NVX_{ij} + \sum_{k \in Z_{ij}}^{K} T_{ij}^{k} \right) \right).$$

For components in which introduction of program redundancy is possible, the following characteristics [9] can be changed:

1. A method of program redundancy implementation: multi-version programming (NVP_{ij}=1, RB_{ij}=0) or recovery block (NVP_{ij}=0, RB_{ij}=1). If the *NVP* method is chosen, then the gene value is established as 0, if the *RB* method is chosen, then the gene value is established as 1.

2. The variant (alternative) number Var_v1 is "failure probability / man-hours". Possible versions are set by the analyst for each component ($1 \le Var_{v1} \le V_{ij}$). V_{ij} is the number of alternative versions of development for each component.

3. The variant number $Var_{v2}...Var_{v10}$ is the probabilities of failure for each version of a component, similar to the above point 2 ($0 \le Var_{v2..10} \le V_{ij}$, 0 – there is no version). The limiting number of program component versions if redundancy will be applied, is set beforehand by the analyst and there cannot be more than 10 versions since a larger quantity brings additional complexity and cost in a component, unjustified by increase in dependability.

If genes $Var_{v2}..Var_{v10}$ take on the value 0, it is considered that program redundancy is not introduced in the given component (B_{ij}=1).

For components in which introduction of program redundancy is not provided, only the version *Var* of the component failure probability and corresponding labor input for achievement of this failure probability changes ($1 \le Var \le$ the number of versions for the given component).

Thus, the phenotype of an individual (decision) is formed from variable characteristics of program components [10]. Table 1 presents the general type of a phenotype with an example of alleles.

Table 1. Phenotype of an individual

Group of components with the possibility of program redundancy									**Group of components with no possibility of program redundancy**		
Component 1				..	Component N				Component 1	..	Component M
NVP/ RB	Var_{v1}	..	Var_{v4}		NVP/ RB	Var_{v1}	..	Var_{v5}	Var		Var
1	3		0		0	1		0	2		3

Crossing of the chosen individuals occurs with the predetermined probability. If parents are not crossed, their cloning takes place.

Crossing occurs by break of parents' chromosome in the set number of points, and descendants receive various attributes of both parents.

In nature there is a huge quantity of attributes and properties of live organisms which are defined by two and more pairs of genes and, on the contrary, one gene frequently supervises many attributes. Besides, action of a gene can be changed by the neighborhood of other genes and environment conditions. In one of studies as early as in 1928 J.A. Filipchenko proved that alongside with the "basic" gene determining an attribute there is a number of genes-modifiers of this attribute. The similar type of inheritance is met frequently. Thus, the phenotype, as a rule, represents the result of complex interaction of genes [11].

Thus, let us introduce the concept "related genes". The related genes are genes whose mutual combination influences a certain attribute of an individual.

Dependability and man-hours for components with possible program redundancy are defined by interacting genes of versions and redundancy method. Breaks of a chromosome can occur both between genes of one program component, and between genes of different components. And the probability of choosing a chromosome break point between genes of one program component (related genes) is predetermined. At uniform cross breeding break can occur either in all points or only between unrelated genes.

The mutation of individuals of a population occurs with the specified probability. In addition to the probability of mutation application to each individual, the probability of mutation application to its each gene is used with the size usually set from 1 up to 10 % [42]. At mutation of binary genes of redundancy method a value invert conversion takes place.

Alleles of genes of alternatives "failure probability / labor input" are characterized by a rank scale in such a manner that each following version is better in relation to dependability, but it is worse in relation to man-hours. Thus, predetermined alternative versions for a specific program component are Pareto optimal. The mutation of such genes can occur in two ways:

1. The choice of any alternative "failure probability / labor input" is equiprobable.

2. The choice of any version "failure probability / labor input" occurs with the probability distributed under the law of normal distribution of probabilities for a discrete random variable [12]. And the choice of the version current value is impossible. At the mutation of genes of versions of program components, for which the value 0 is possible (absence of any version), such value is added as an additional possible value. Table 3 presents probabilities of versions' choice at mutation of a gene of the program component version, whose current value is equal to 4, and the total number of specified versions is equal to 7.

Table 3. Probability distribution of a version choice

Absence of the version	Var. 1	Var. 2	Var. 3	Var. 4 (current)	Var. 5	Var. 6	Var. 7
0.015625	0.09375	0.234375	0.3125	0	0.234375	0.09375	0.015625

The genetic algorithm is based on ideas of a method with independent Shaffer's selection for multi-criterion optimization *VEGA* (*Vector Evaluated Genetic Algorithm*) [13].

Selection occurs with the probability proportional to criterion value. The probability of an individual to be selected under criterion *S* is calculated by the formula [14]:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} = \frac{f_i}{\overline{f} \cdot N},$$

where f_i is the suitability of the individual *i*; $\overline{f} = \frac{1}{N}\sum_{j=1}^{N} f_i$ is the average suitability of the population; *N* is the size of the population; $\sum_{j=1}^{N} p_j = 1.$

The probability of an individual to be selected under minimized criterion T_s is calculated by the following formula [15]:

$$p_i = \frac{-f_i + C}{N \cdot C - \sum_{j=1}^{N} f_j},$$

where f_i is the suitability of the individual i; *C* is the constant determining the minimal suitability of the $C : p_i \geq 0, \forall i$ population; N is the size of the population; $\sum_{j=1}^{N} p_j = 1.$

In the considered algorithm the constant C is equal to the maximal suitability of an individual in population. Thus, the minimal probability of an individual to be the selected under criterion T_s is equal to 0.

When defining the suitability of an individual, the values *S* and T_s are calculated under the algorithm represented in fig. 1.

Each individual with calculated criteria is recorded into data array. Search of similar calculated earlier individual is carried out before criteria calculation for the following individual [10]. This action is reasonable and allows saving an operating time of algorithm since calculation of criteria is much longer than search of an identical individual in data array of decisions.

For the solution of the multi-criterion task of conditional optimization, it is necessary to use the approach based on conditional task reduction to the unconditional one. Search of Pareto optimal solutions is carried out under the design of *VEGA* method. Task solving with restrictions should belong to a Pareto set, as well as it should be in a legitimate range. Therefore, in addition to the offered algorithm, we should introduce an additional procedure that allows us to bring solutions into a legitimate range [14].

To make it possible to solve a conditional optimization task, each restriction should be considered as a separate criterion function and consequently, an initially conditional task with several criterion functions is reduced to the unconditional task of optimization. The transformation of an initial task of conditional multi-criterion optimization has the following form [14]:

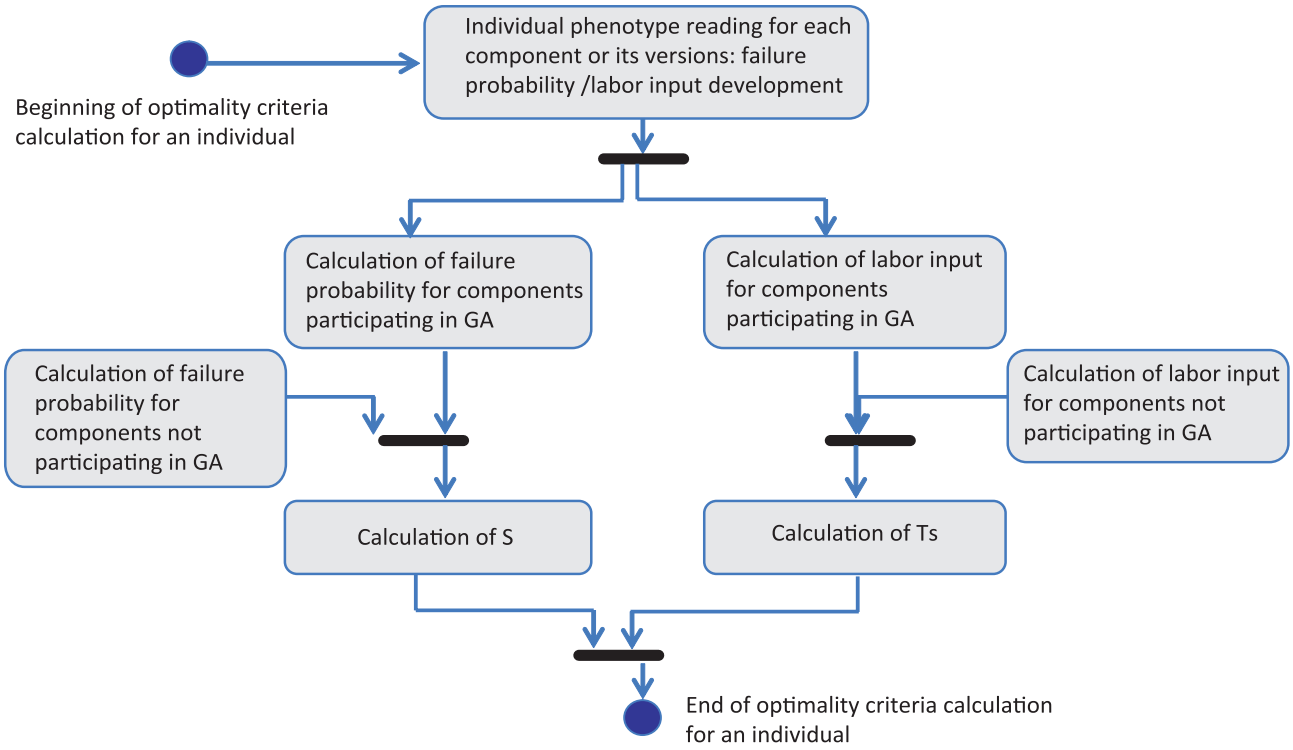

Fig. 1. Algorithm of optimality criteria calculation

Criterion functions of an initial task are $F(X) \rightarrow opt$, restrictions of the initial task are $G(X) \leq B$.

Criterion functions of the transformed task are $F(X) \rightarrow opt$, restrictions of the transformed task are $|G(X) - B| \rightarrow opt$.

For the first several iterations of the algorithm, the initial conditional task is solved, but without taking into account restrictions. Next, to receive a greater number of solutions belonging to the legitimate range, search proceeds this time without taking into account criterion functions of the initial task, but restrictions only. Thus, search of solutions is made only as to functions – restrictions, which brings the most part of the population into a legitimate range, but with loss of solutions' quality under criteria of optimization [14].

The genetic algorithm for multi-criterion conditional optimization of program architecture [16] is also based on ideas of *VEGA* method (*Vector Evaluated Genetic Algorithm*) with independent Shaffer's selection at multi-criterion optimization.

The difference of *VEGA* algorithm from *GA* (genetic algorithm) of unconditional optimization consists in the fact that each restriction in it is considered as additional criterion of optimization. With the part of generations, the algorithm works without taking into account additional criteria of optimization, and then with the rest of generations the algorithm works as to two criteria of restriction violation on S, T_s, and criteria of restriction violation RT_{ij}.

Input parameters of *GA* are the following:

- population size (N);
- probability of crossing (*prob_cross*);
- type of crossing (1,2,3-point, uniform);
- probability of related genes break (*prob_cross_inter*);
- probability of individual mutation (*prob_mutate*);
- probability of gene mutation (*prob_mutate_gen*);
- halt criteria (the maximal operating time *time_ga*, the number of populations without solution improvement *stagnancy*, the number of populations *pop_count*);

- percent of populations for processing of restrictions *percent_bound*;
- number of restrictions for the period of components' execution *B*.

The algorithm is implemented by the following sequence of actions:

1. Generation of parental population *P* with size *N* of casual individuals.
2. Calculation of criteria for all individuals of population *P*.
3. Proportional selection *N*/2 of individuals from *P* under criterion *S* in intermediate population *P'*.
4. Proportional selection N/2 of individuals by criterion T_s in intermediate population *P'*.
5. Crossing with probability *prob_cross N*/2 of casually chosen pairs of individuals from intermediate population *P'*. Generation of basic population *P* from *N* chosen individuals.
6. Execution of the mutation operator with the probability *prob_mutate* on each individual of basic population *P* and on each gene of an individual with the probability *rob_mutate_gen.*
7. Calculation of optimization criteria values for all individuals of population *P*.
8. Selection from population *P* of undominated solutions. If there are solutions in the decisions found earlier which dominate them, then stagnancy = *stagnancy+1*.
9. If even one criterion of halt works, then the algorithm stops.
10. If the number of population is less, or equal to *percent_bound*pop_count,* then move to step 3.
11. Proportional selection of *N* / (2+*Q*) of individuals from *P* under criterion of restriction violation on *S* in intermediate population *P'*.
12. Proportional selection of *N* / (2+*Q*) of individuals under criterion of restriction violation on T_s in intermediate population *P'*.
13. Proportional selection *N* / (2+*Q*) of individuals under each criterion of restriction violation for the period of RT_{ij} component implementation in intermediate population *P'*.
14. Crossing with probability *prob_cross N* / (2+*Q*) of casually chosen pairs of individuals from intermediate population *P'*. Generation of basic population *P* from *N* chosen individuals.
15. Execution of the mutation operator with the probability *prob_mutate* on each individual of basic population *P* and each gene of an individual with the probability *prob_mutate_gen.*
16. Calculation of criteria under restrictions for all individuals of population *P*.
17. Selection from population *P* of the best decisions under criteria of restrictions. If there are better solutions in the decisions found earlier, then stagnancy = *stagnancy+1*.
18. If even one criterion of halt works, then the algorithm stops, otherwise move to step 11.

Undominated decisions obtained for each iteration are selected in a Pareto set. Decisions of a Pareto set cannot be preferred to each other, therefore after its generation the task can be considered mathematically solved [17].

The developed genetic algorithm has been tested on specially prepared tasks and has also been used in development of the software for corporate control systems and OSW. With the help of the developed GA, all tasks have been effectively solved, efficient architectural decisions have been found.

References

1. **Kovalev I.** Optimal Time Cyclograms of Spacecrafts Control Systems / **I. Kovalev, O. Davydenko** // Advances in Modeling and Analysis. Vol.48. # 2-3.1996. AMSE PRESS. P. 19-23.

2. **Tsarev R.Yu.** Firmware of fail-safe and catastrophe-safe control and of the information processing systems: the study / **A.V.Anikonov, M.Yu..Slobodin, R.Yu.Tsarev.** – M.: Макс-press, 2006. – 244 p.

3. **Tsarev R.Yu.** Optimization and simulation approach to synthesis of the automated control systems / **I.V.Kovalev, M.V.Tjupkin, R.Yu.Tsarev, Yu.D.Tsvetkov** // Software products and systems. – 2007. – # 3. – pp. 73-74.

4. **Tsarev R.Yu.** System of multi-attributive formation of multi-version software in fail-safe control systems: PhD Thesis: Krasnoyarsk, 2003. – 143 p.

5. **Novoy A.B.** System for the analysis of software architectural dependability: PhD Thesis: Krasnoyarsk, 2011 – 131 p.

6. **Sheenok D.A., Kukartsev V.V.** Costs estimate for upgrading software of dependability critical systems // SibSAU Bulletin. Issue 5 (45). – 2012. – pp. 62-65.

7. **Sheenok D.A.** Updating of program architecture model for multi-criterion conditional optimization // Prospects of development of information technologies: Matter of XI international scientific – practical conference. – 2013. – pp 76-81.

8. **Rusakov M.A.** Multistage analysis of architectural dependability in complex information – management systems: PhD Thesis: Krasnoyarsk, 2005 – 168 p.

9. **Sheenok D.A., Kukartsev V.V.** Program architecture optimization of logistical information systems // Logistical systems in global economy: Matters of international scientific-practical conference. (March, 14-15, 2013, Krasnoyarsk). Part 1. – 2013. – pp 138-145.

10. **Sheenok D.A.** Genetic algorithm of software optimum architecture search // Urgent issues of modern science: Matter of IV international scientific conference on December, 14-15, 2012 – pp. 62-68.

11. **Inge-Vechtomov S.G.** Genetics with fundamentals of selection: Textbook for the biological special subject in universities. – M.: Higher school, 1989. – 591 p.

12. **Spitsyn V.G., Tsoy Yu. R.**Rrepresentation of knowledge in information systems: tutorial manual. – Tomsk: Publishing house TPU, 2006. – 146 p.

13. **Sergienko R.B.** Coevolution algorithm for tasks of conditional and multi-criterion optimization / **R.B.Sergienko, E.S.Semenkin** // Software products and systems. – # 4. – 2010. – pp. 24-28.

14. **Gumennikova A. V.** Adaptive search algorithms for the solution of complex tasks of multi-criterion optimization: PhD Thesis: Krasnoyarsk, 2006 – 129.

15. **Sopov E. A.** Evolutionary algorithms of complex systems modeling and optimization: PhD Thesis: Krasnoyarsk, 2004 – 129 p.

16. **Sheenok D.A.** Tool for designing of optimum architecture of fail-safe program systems // Program engineering. # 6. – 2013. – P. 20-26.

17. **Sergienko R.B.** Automated generation of fuzzy qualifiers by self-adapting coevolution algorithms: PhD Thesis: Krasnoyarsk, 2010 – 192 p.