An Algorithm for Updating the Electronic Structure of a Product Based on Evolutionary Programming with Non-Dominated Sorting

Gayk A. Gabrielyan

Abstract—**The relevance of this work stems from the regular need to revise the composition of complex technical products during their life cycle. Replacing discontinued components, updating specifications, and accounting for budget constraints require an automated search for compromise solutions instead of labor-intensive and error-prone manual scanning of options. The aim of this work is to develop an algorithm for the multi-criteria synthesis of a sequence of operations for editing the electronic structure of a product (ESP), allowing one to obtain from the original configuration a new one close to the target ESP based on a set of criteria for improving characteristics, completeness of assembly, and cost non-exceedance. The scientific novelty lies in the development of the specialized algorithm for evolutionary programming of electronic structure of a product – a specialized version of NSGA-II in which the crossover operator is replaced by a mutation operator operating directly on the ESP tree and the sequence of ESP tree editing instructions, as well as in the post-processing procedure, including the removal of redundant instructions and ranking of Pareto-optimal solutions using the TOPSIS method with entropy weights. In addition, a specialized version of the MOEA/D algorithm, adapted to the same representation of ESP editing programs, was developed and compared with the proposed specialized NSGA-II. The conducted numerical experiment which involved automated reconfiguration of computational assemblies using the specialized algorithm demonstrated the algorithm's convergence with respect to the hypervolume metric, the algorithm generated 17 non-dominated Pareto-front solutions. The algorithm for redundant instruction removal reduced the number of instructions in the ESP tree editing programs belonging to the constructed Pareto-front by an average of 32% of the original number of instructions. The comparison with the specialized MOEA/D showed that the specialized NSGA-II achieves a higher hypervolume value under matched computational budgets, with the advantage most pronounced for small population sizes (up to 32% relative gain at n=10) and diminishing as the population size grows (about 3% at n=100). The practical significance of the work is that the proposed approach automates the task of reconfiguring products using a limited set of components: instead of manually exploring the space of all available configurations, the designer receives a ranked list of specific ESP editing programs with their numerical scores for each criterion. The method is applicable in product lifecycle management (PLM/PDM) systems to support decision-making when updating product specifications and when planning maintenance or component replacement.**

Manuscript received August 17, 2026.

Gayk A. Gabrielyan, Lecturer of the Department of Corporate Information Systems, Institute of Information Technologies, MIREA – Russian Technological University, 78, Vernadsky pr., Moscow, 119454 (email: gabrielyan@mirea.ru).

Keywords—**electronic structure of the product, evolutionary programming, genetic algorithms, multi-criteria optimization, NSGA-II, MOEA/D, Pareto front, TOPSIS**

I. Introduction

An electronic structure of a product (ESP) is a computer model of a product that describes the product as a set of its structural elements and the relationships between them [1]. ESP is an electronic design document intended for use in a computing environment and for organizing information interaction between automated systems. ESP is used alongside related forms of electronic product descriptions regulated by the regulatory framework of the unified system of design documentation [2, 3].

ESP is a key data object in Product Lifecycle Management (PLM) and in Product Data Management (PDM) systems. ESP is used in mechanical engineering, instrument making, aerospace, electronics, and other areas where control over the composition of a product is required at all stages of its life cycle, from design and production to maintenance and disposal [4].

During the product life cycle, the ESP is regularly updated: components are replaced with similar components or discontinued, and their specifications are updated, which is consistent with the general provisions of the product development and launch system [5]. The objective of ESP updating is to automatically construct, based on the existing ESP and the target ESP, a sequence of structural changes that bring the original ESP closer to the target ESP, while limiting the set of available ESP components. Manually solving this problem for complex products is labor-intensive and error-prone, which justifies the need to develop automated methods for its solution.

Existing approaches to automating the synthesis and updating of ESP can be divided into two categories. The first category is based on large language models (LLM): a method for synthesizing ESP based on language models, constraint programming, and intermediate representation translators has been proposed in [6], another method for updating ESP based on an event-driven approach has been proposed in [7], which uses a loop of generating and verifying LLM proposals based on ERP system events. Such methods as [6, 7] are effective in the presence of natural-language descriptions of the required ESP changes; however, their results depend on the quality of LLM generation results and require subsequent verification for compliance with integrity constraints. The second category,

which the algorithm proposed in this paper belongs to, is based on evolutionary computation and does not require the use of an LLM, forming a solution through an explicit multi-criteria search in the space of ESP editing programs.

Evolutionary algorithms represent a class of metaheuristic optimization methods inspired by the principles of biological evolution: selection, mutation, crossover, and inheritance of traits. Their distinctive feature is the ability to effectively explore large spaces of possible solutions without using information about the gradient of the objective function, which makes them applicable to discrete, combinatorial, and structural optimization problems. In multi-criteria optimization problems, where improving a solution with respect to one criterion may lead to a deterioration of the solution with respect to another criterion, the goal is to find not a single solution, but a set of solutions such that none of them dominates the others simultaneously with respect to all criteria (the set of Pareto-optimal solutions) [8]. Multi-criteria formulation of optimization problems for technological objects is widespread in engineering practice. A systematic review of scientific publications for 2013–2023 shows a steady increase in the interest in this issue and a variety of applied methods – from classical criteria convolution methods to modern evolutionary algorithms [9].

Evolutionary programming is a separate branch of evolutionary computation in which programs or behavioral strategies evolve rather than sets of numerical parameters [10]. Unlike genetic programming, which operates on tree-like expressions and actively uses the subtree crossover operator [11], evolutionary programming emphasizes the mutation operator as the primary mechanism for generating new solutions and may not use crossover at all, which simplifies maintaining the syntactic and semantic correctness of evolving programs.

NSGA-II (Non-dominated Sorting Genetic Algorithm II) is one of the most widely used multi-criteria evolutionary optimization algorithms [12]. The algorithm uses a fast non-dominated sorting procedure to rank the solutions of a population into layers (fronts) of non-dominated solutions and calculates a crowding metric to maintain population diversity within each front without involving additional hyperparameters, which was a key improvement over earlier multi-criteria optimization algorithm.

NSGA-II has been successfully applied to design and optimization problems of technical systems of various nature: in particular, its effectiveness has been demonstrated in comparison with the MOGA algorithm on the problem of optimizing the shape of a hydraulic turbine runner [13]. Specialized versions of NSGA-II adapt its mechanisms to specific types of solution spaces – discrete, combinatorial, hierarchical. An example of such an adaptation is the improved integer version of NSGA-II for solving the generalized assignment problem [14].

An alternative family of multi-criteria evolutionary algorithms is based on the decomposition of a multi-criteria problem into a set of scalar subproblems, each corresponding to a distinct weight vector in the criteria space. MOEA/D (Multiobjective Evolutionary Algorithm based on Decomposition) is the most widely used representative of this family [20]. Instead of ranking the population into non-dominated fronts as in NSGA-II, MOEA/D maintains a fixed set of weight vectors, defines a neighborhood relation between subproblems based on the distance between their weight vectors, and optimizes each subproblem using information exchanged only within its neighborhood, typically via the Chebyshev scalarization function.

In this paper, we propose the specialized algorithm for evolutionary programming of electronic structure of a product – a specialization of NSGA-II for the problem of synthesizing programs that edit hierarchical ESP structures. The algorithm replaces the crossover operator with a specialized mutation operator that operates directly on the ESP tree and the sequence of ESP editing instructions. We supplement the algorithm with post-processing of the ESP editing sequence by removing redundant instructions and ranking the resulting solution set using the TOPSIS method.

Given the widespread use of MOEA/D as a baseline for multi-criteria evolutionary algorithms, in this paper we additionally develop a specialized version of MOEA/D adapted to the same representation of ESP editing programs and compare the two algorithms on the ESP updating problem.

II. Methods and Algorithms

A. Representation of ESP as a Tree

In the problem under consideration, the ESP is represented as a simplified tree: the hierarchical structure of the ESP components is expressed through parent-child relationships across connectors, while the attributes/characteristics of each ESP component are stored separately and used to calculate the solution suitability criteria. The ESP tree t is represented as a set of pairs:

$$t = \{(\mathrm{id}, i), (s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}, \quad (1)$$

where i is the identifier of the ESP tree t, the pair (id, i) being intended to store the identifier of the ESP tree t; s_1, s_2 and s_k are the connectors of the ESP tree t to which the child ESP trees t_1, t_2 and t_k are connected, respectively; k is the number of connectors in the ESP tree t.

Each connector s_l can be in 1 of the 2 states: occupied – $(s_l, t_l) \in t$ when $t_l \neq \varnothing$, or free – $(s_l, \varnothing) \in t$.

An example of an ESP tree with one child subtree:

$$t_{example} = \{(\mathrm{id}, i_1), (s_1, \{(\mathrm{id}, i_2), (s_3, \varnothing), (s_4, \varnothing)\}), (s_2, \varnothing)\}. \quad (2)$$

In (2) the root component i_1 has two connectors: the ESP component i_2 (with 2 free connectors s_3 and s_4) is connected to connector s_1, and connector s_2 is free. This structure directly corresponds to the hierarchy of the product composition. For example, the motherboard can serve as the root of an ESP subtree, to which the processor, memory modules, and storage devices are connected, and the replacement of one or more ESP components can be performed by the editing program ρ. An example of a real

ESP of a computer assembly, used further in the experimental part, is shown in Figure 1.

B. Basic Operations on the ESP Tree

Each linear sequence of ESP tree editing operations $\rho \in \mathbb{P}$, where $\mathbb{P}$ is the set of all possible sequences of ESP editing operations, may include instructions of two types.

The ESP component connection instruction of the form $(\text{connect}, i_p, s_p, t_c)$ connects the ESP component tree t_c to the free connector s_p of the ESP subtree with identifier i_p. The algorithm for applying an instruction of this type to the ESP tree t performs a recursive traversal of the ESP tree t and replaces the pair $(s_p, \varnothing)$ with (s_p, t_c) in the ESP subtree with identifier i_p. The computational complexity of this operation is $O(\eta)$, where η is the number of nodes in the ESP tree t to which the instruction is applied.

The connector release instruction of the form (free, i_p, s_p) disconnects the ESP subtree connected to connector s_p of the ESP subtree with identifier i_p, replacing the pair (s_p, t_c) with $(s_p, \varnothing)$ in the ESP tree t. The computational complexity of this operation is $O(\eta)$, where η is the number of nodes in the ESP tree t to which the instruction is applied.

The function $\text{app}(t, \rho)$ applies the ESP editing program ρ to the ESP tree t, sequentially executing each instruction contained in the sequence of editing operations ρ. Thus, the computational complexity of $\text{app}(t, \rho)$ is $O(\eta|\rho|)$. An example of an ESP tree obtained by applying an editing program to the tree shown in Figure 1 is given in Figure 2 in the experimental part.

C. New ESP Synthesis Based on a Given ESP

Let there be given an initial ESP t and a target ESP t_{target}, which is unavailable for assembly from the set of available ESP components B due to the absence of some components of the initial ESP in B. It is required to find a sequence of editing operations of the initial ESP $\rho \in \mathbb{P}$, such that the ESP $\text{app}(t, \rho)$, obtained from the initial ESP t by applying the sequence of editing operations ρ, would be no worse than the target ESP t_{target} with respect to a set of 3 criteria:

$$\begin{cases} \arg\max\limits_{\rho \in \mathrm{P}} \text{improvement}(t, \rho, t_{\text{target}}), \\ \arg\max\limits_{\rho \in \mathrm{P}} \text{completeness}(t, \rho, t_{\text{target}}), \\ \arg\min\limits_{\rho \in \mathrm{P}} \text{overprice}(t, \rho, t_{\text{target}}). \end{cases} \tag{3}$$

The characteristics improvement criterion in (3) gives a numerical estimate of how much the characteristics of the components of the synthesized ESP exceed the characteristics of the components of the target ESP:

$$\text{improvement}(t, \rho, t_{\text{target}}) = \\ = \frac{1}{2}\left(1 + \frac{1}{|T|} \sum_{i \in V(\text{app}(t,\rho))} \sum_{j \in V(t_{\text{target}})} \text{score}(i, j)\right), \tag{4}$$

where $T = \{\tau(i) : i \in V(\text{app}(t, \rho))\}$ is the set of types of ESP components of the synthesized ESP, the type of an ESP component corresponds to its functional purpose; V is a mapping that returns the components of a tree; τ maps an ESP component to its type.

The mapping $\text{score}(i, j)$ in (4) returns 0 if the types of ESP components i and j are different, and the average over the results of all attributes comparison otherwise:

$$\text{score}_{\text{attr}}(a_k^i, a_k^j) = \begin{cases} 1, & a_k^i \succ a_k^j, \\ 0, & a_k^i = a_k^j, \\ -1, & a_k^i \prec a_k^j, \end{cases} \tag{5}$$

where a_k^i is the value of the k-th attribute of the i-th ESP component, a_k^j is the value of the k-th attribute of the j-th ESP component.

The relations $\succ$ and $\prec$ in (5) are given by a partial order on attribute values, specific to the type of ESP component (for example, a higher CPU clock speed is better, a lower TDP value is better). The value of the improvement criterion (4) belongs to the range [0, 1].

The assembly completeness criterion in (3) estimates the proportion of the target ESP's component types that are present in the synthesized ESP:

$$\text{completeness}(t, \rho, t_{\text{target}}) = 1 - \frac{|T_{\text{target}} - T|}{|T_{\text{target}}|}, \tag{6}$$

where $T_{\text{target}} = \{\tau(j) : j \in V(t_{\text{target}})\}$ is the set of types of ESP components of the target ESP; a value of 1 for criterion (6) corresponds to the presence of all types of ESP components of the target ESP in the synthesized ESP. The value of the completeness criterion (6) belongs to the range [0, 1].

The cost overrun criterion, overprice in (3), estimates the relative excess of the price of the synthesized ESP over the price of the target ESP:

$$\text{overprice}(t, \rho, t_{\text{target}}) = \\ = \frac{\max\left(\sum\limits_{i \in V(\text{app}(t,\rho))} \text{price}(i) - \sum\limits_{j \in V(t_{\text{target}})} \text{price}(j), 0\right)}{\sum\limits_{i \in V(\text{app}(t,\rho))} \text{price}(i)}, \tag{7}$$

where $\text{price}(i)$ is the cost of the ESP component.

Criterion (7) equals zero when the synthesized ESP is no more expensive than the target ESP, and increases proportionally to the relative price increase. The value of criterion (7) is also limited to the range [0, 1].

Thus, all three criteria (4) – (7) are defined on a single scale [0, 1]. The set of criteria covers three engineering requirements, possibly mutually contradictory, typical for the task of reconfiguring a complex product, and provides a meaningful multi-criteria formulation of problem (3).

D. Complexity of Exhaustive Search

If the number of instructions $|\rho|$ in the sequences of ESP editing operations ρ from the set $\mathbb{P}$ is equal to μ, and the number of ESP editing operations required to transform the original ESP into the target ESP is estimated as $\eta \cdot |B|$, where η is the number of nodes in the target ESP tree and B is the set of available ESP components, the cardinality of the set $\mathbb{P}$ is equal to $(\eta \cdot |B|)^{\mu}$.

Since the length of the ESP editing program $\rho \in \mathbb{P}$ is not limited in practice and depends on the differences between the original ESP and the target ESP, exhaustive search is difficult even for small sizes of the ESP tree and the set of ESP components B, which makes it reasonable to apply optimization methods based on evolutionary programming as a practically feasible approach to problem (3).

E. The Algorithm for ESP Updating

To solve the stated problem, we propose an evolutionary programming algorithm with non-dominated sorting – a specialized version of the NSGA-II algorithm [12], in which solutions are represented by sequences of ESP editing operations, the crossover operator is absent, and mutation is the only mechanism for generating new solutions, which is characteristic of evolutionary programming [10].

Algorithm 1 – Specialized NSGA-II for evolutionary programming of electronic structure of a product

```
1.  function ESP_UPDATE(t, F, B, n, m, r)
2.      h ← 0
3.      Q_h ← ∅
4.      P_h ← {ρ_1^h, ρ_2^h, …, ρ_n^h}
5.      repeat m times
6.          R_h ← P_h ∪ Q_h
7.          F ← FAST_NON_DOMINATED_SORT(R_h, F)
8.          P_{h+1} ← ∅
9.          i ← 1
10.         while |P_{h+1}| + |F_i| < n do
11.             P_{h+1} ← P_{h+1} ∪ F_i
12.             i ← i + 1
13.         C_i ← CROWDING_DISTANCES(F_i, F)
14.         F_i ← SORT(F_i, C_i, desc)
15.         for j ∈ {1, …, n − |P_{h+1}|} do
16.             P_{h+1} ← P_{h+1} ∪ {F_ij}
17.         Q_{h+1} ← ∅
18.         for ρ ∈ SELECT(P_{h+1}, F) do
19.             if UNIFORM(0,1) ≤ r then
20.                 ρ ← MUTATE_OPS(t, ρ, B)
21.             Q_{h+1} ← Q_{h+1} ∪ {ρ}
22.         h ← h + 1
23.     A ← {MINIMIZE_OPS(t, ρ): ρ ∈ F_0}
24.     return A
```

Algorithm 1 takes as input the initial ESP tree t, the set of criteria F, the set of available ESP components B, and the hyperparameters n, m, r, and returns the set of ESP editing programs A (see line 24).

Lines 1-4 perform initialization: the iteration number h is set to zero, the offspring population Q_h is assumed empty, and the initial population P_h consists of n empty ESP editing programs. The main loop of the algorithm (see lines 5-22) is repeated m times. Line 6 forms the combined population R_h of solutions of the current generation and the offspring generation. The fast-non-dominated-sort function (see line 7) sorts the solutions R_h into layers of non-dominated solutions $\mathcal{F}$ based on the values of the criteria in the set F, as in the original NSGA-II algorithm [13].

Lines 8-12 form the new population P_{h+1}: layers $\mathcal{F}_i$ are added to it entirely as long as the next layer does not exceed the remaining capacity of the population n. The crowding-distances function (see line 13) computes the crowding estimates C_i of the solutions of layer $\mathcal{F}_i$ based on the values of the criteria F, after which the sort function (see line 14) orders the solutions of layer $\mathcal{F}_i$ in descending order of crowding, and the remaining places in population P_{h+1} are filled with the least crowded solutions (see lines 15-16).

Lines 17-21 form the new offspring population Q_{h+1}: the "select" function (see line 18) implements tournament selection of candidates from population P_{h+1} based on the values of criteria F; the "uniform" function (see line 19) returns a random real number uniformly distributed on the interval $[0,1]$, and with probability r the mutation operator mutate-ops (see line 20) is applied to the selected candidate.

At the final step (see line 23), the redundant instruction removal algorithm "minimize-ops" is applied to the solutions of the best layer $\mathcal{F}_0$, the result of which is returned as the set of optimal ESP editing programs, which is subsequently ranked using the TOPSIS method.

F. Mutation Operator of ESP Editing Operations

The mutation operator is implemented as follows: with equal probability, either the last instruction is removed from the ESP editing program ρ, or a new instruction is added to the end of the program. Before generating new instructions, the current ESP editing program ρ is sequentially applied to the ESP tree t, obtaining the current state of the ESP tree $t' = \text{app}(t, \rho)$. The new instruction generation procedure used in the mutation operator is defined by Algorithm 2.

Algorithm 2 – ESP mutating instruction generation

```
1.  function MUTATION(t, B)
2.      C ← CONNECTED_SOCKETS(t)
3.      F ← FREE_SOCKETS(t)
4.      loop
5.          r ← RAND({0,1})
6.          if r = 0 ∧ C ≠ ∅ then
7.              i_p, s_p ← RAND(C)
8.              return (FREE, i_p, s_p)
9.          if r = 1 ∧ F ≠ ∅ then
10.             i_p, s_p ← RAND(F)
11.             if s_p ∈ B then
12.                 t_c ← RAND(B[s_p])
13.                 return (CONNECT, i_p, s_p, t_c)
```

Algorithm 2 takes the ESP tree t as input and the set of ESP components available for connection B.

The "connected_sockets" function (see line 2) returns the set of pairs (i_p, s_p) corresponding to the occupied connectors of the ESP tree t; the "free_sockets" function (see line 3) returns the set of pairs (i_p, s_p) corresponding to the free connectors of the ESP tree t.

The "rand" function (see lines 5, 7, 12) returns a random element of the set passed to it: at line 5 "rand" generates a

random bit determining the type of the instruction to be generated; at line 7 – a randomly chosen occupied connector from the set C; at line 12 – a randomly chosen free connector from the set F. At line 13, it is checked whether the set of available ESP components B contains at least one compatible component $B[s_p]$ for connector s_p; if so, at line 14 the "rand" function returns a randomly chosen compatible component t_c from the set $B[s_p]$. If a suitable instruction is not formed on the current iteration of the loop, the attempt is repeated (see loop, lines 4-16). The mutation operator of the sequence of operations "mutate-ops", used in the main loop of the Algorithm 1, is defined by Algorithm 3.

Algorithm 3 – Mutation of a single ESP editing program

```
1.  function MUTATE_OPS(t, ρ, B)
2.      if RAND({0,1}) ∧ |ρ| > 0 then
3.          return POP(ρ)
4.      t' ← APP(t, ρ)
5.      o ← MUTATION(t', B)
6.      return PUSH(ρ, o))
```

Algorithm 3 with a probability determined by the random bit at line 2, removes the last instruction of the ESP editing program ρ using the pop function (see line 3); otherwise, the program ρ is applied to the ESP tree t (see line 5), the "mutation" function (see Algorithm 2) generates a new instruction o for the current state of the ESP tree t' (see line 6), and the "push" function (see line 7) adds instruction o to the end of the ESP editing program ρ.

The instruction generated by Algorithm 3 is guaranteed to change the structure of the ESP tree. Since Algorithm 1 has no crossover operator, mutation is the only source of diversity, which at a high probability ensures intensive exploration of the space of possible ESP editing programs, similar to the role of mutation in classical evolutionary programming schemes [10].

G. Minimizing ESP Editing Operations

After evolutionary search, ESP editing programs ρ typically contain redundant instructions – pairs that connect an ESP component and then disconnect it, without affecting the resulting ESP tree. A specialized instruction minimization algorithm removes such instructions in two stages.

On the first stage Algorithm 4 identifies redundant instructions. Algorithm 4 traverses the ESP editing program ρ from left to right, maintaining a table M mapping a connector (i_p, s_p) to the number of the last "connect" instruction for that connector. When a "free" instruction is encountered for a connector present in M, both numbers ("connect" and "free") are added to the set of redundant instruction numbers R.

Algorithm 4 – Redundant instruction identification

```
1.  function FIND_REDUNDANT_OPS(t, ρ)
2.      R ← ∅, M ← ∅
3.      for o_j ∈ ρ, j ∈ {1, …, |ρ|} do
4.          match o_j
5.              case (CONNECT, i_p, s_p, t_c)
6.                  M[i_p, s_p] ← j
7.              case (FREE, i_p, s_p) ∧ (i_p, s_p) ∈ M
8.                  R ← R ∪ {j, M[i_p, s_p]}
9.                  delete M[i_p, s_p]
10.     return R
```

Algorithm 4 takes as input the ESP tree t and the ESP editing program ρ, and returns the set of redundant instruction numbers R. Each instruction o_j of program ρ is matched against one of two patterns (see lines 4-12): if instruction o_j is an ESP component connection instruction, the number j is written into table M under the key of connector (i_p, s_p) (see line 6); if instruction o_j is a connector release instruction for which table M already contains an entry, both numbers – the current j and the previously stored $M[i_p, s_p]$ – are added to the set R, and the entry is removed from table M (see lines 9-10).

On the second stage instructions are filtered. A new ESP editing program ρ' is formed, containing only instructions $o_j \in \rho$ which numbers are not in R and which actually change the ESP tree ($t' \neq t$ after application). The first condition removes connect/free pairs for a single connector, the second removes intermediate instructions on ESP components that are subsequently disconnected.

Algorithm 5 – Redundant instruction removal

```
1.  function MINIMIZE_OPS(t, ρ)
2.      R ← FIND_REDUNDANT_OPS(t, ρ)
3.      ρ' ← ∅
4.      for o_j ∈ ρ, j ∈ {1, …, |ρ|} do
5.          t' ← APP(t, {o_j})
6.          if j ∉ R ∧ t ≠ t' then
7.              ρ' ← PUSH(ρ', o_j)
8.              t ← t'
9.      return ρ'
```

Algorithm 5 takes the ESP tree t and the ESP editing program ρ as input, calls the "find-redundant-ops" function (see Algorithm 4) to obtain the set of redundant instruction numbers R (see line 2), and sequentially iterates over the instructions o_j of program ρ (see lines 4-9): instruction o_j is added to the resulting program ρ' by the "push" function only if its number j is not in the set R and its application actually changes the current ESP tree t (see lines 6-8). As a result, the ESP editing program is shortened without changing the final state of the ESP tree. The computational complexity of Algorithm 5 is $O(|\rho| \cdot \eta)$.

H. Ranking Solutions Using the TOPSIS Method

To select one or more preferred solutions from the Pareto front, the TOPSIS method (Technique for Order Preference by Similarity to Ideal Solution) is applied [15]. The method is based on computing the distance of each solution to the ideal point (best values for all criteria) and to the anti-ideal point (worst values) in the criteria space, and subsequently ranking solutions by relative closeness to the ideal.

The input to the TOPSIS method is the matrix $M = (m_{ij})_{n\times 3}$, where n is the number of solutions in the found Pareto front to be ranked by the 3 criteria (4)–(7), and the value m_{ij} corresponds to the numerical score of the i-th solution with respect to the j-th criterion.

The matrix M is normalized and weighed:

$$m_{ij}^{*} = \frac{w_j m_{ij}}{\sqrt{\sum_{k=1}^{n} m_{kj}^2}}, \quad (8)$$

where w_j is the weight of the j-th criterion; the weights of criteria (4) – (7), $\vec{w} = (w_1, w_2, w_3)$, are computed using the entropy weight method based on the distribution of criterion values in the current Pareto front: a criterion whose values are distributed more evenly across the front's solutions (has higher entropy) carries less discriminating information and receives a smaller weight. A criterion with larger variation of values, on the contrary, receives a larger weight.

For the matrix M, the entropy of the j-th criterion is computed as:

$$E_j = -\frac{1}{\ln(n)} \sum_{i=1}^{n} \left(\frac{m_{ij}}{\sum_{k=1}^{n} m_{kj}} \ln \left(\frac{m_{ij}}{\sum_{k=1}^{n} m_{kj}} \right) \right), \quad (9)$$

and the resulting weight of the j-th criterion is computed as:

$$w_j = \frac{1 - E_j}{\sum_{l=1}^{3} E_l}. \quad (10)$$

The weighted values from the criteria matrix are compared to the ideal point $z^{+} = (z_1^{+}, z_2^{+}, z_3^{+})$ and to the anti-ideal point $z^{-} = (z_1^{-}, z_2^{-}, z_3^{-})$, given as fixed reference values corresponding to the theoretically best and worst combination of the values of criteria (4) – (7).

For each solution, an index of its closeness to the ideal solution is computed:

$$c_i = \frac{d_i^{-}}{d_i^{+} + d_i^{-}}, \quad (11)$$

$$d_i^{\pm} = \sqrt{\sum_{j=1}^{3} \left(m_{ij}^{*} - z_j^{\pm} \right)^2}, \quad (12)$$

where d_i^{+} and d_i^{-} are the Euclidean distances of the i-th solution to the ideal point z^{+} and to the anti-ideal point z^{-}; m_{ij}^{*} corresponds to the normalized and weighted (8) numerical score of the i-th solution with respect to the j-th criterion; c_i is the closeness index of the i-th solution to the ideal solution. The solutions in the Pareto front found at the previous stage are ranked in decreasing order of the computed c_i values; the solution with a value of c_i close to one is the most preferable; the solution with the highest value of c_i is assigned TOPSIS rank 0.

I. Specialized MOEA/D for Evolutionary Programming of Electronic Structure of a Product

To provide a comparison baseline for the proposed specialized NSGA-II, we additionally developed a specialized version of MOEA/D [16] adapted to the same representation of ESP editing programs.

Each subproblem is scalarized using the Chebyshev approach: a weight vector assigns a relative importance to each of the three criteria (4)–(7), and the scalar value of a subproblem is computed as the largest of the three weighted deviations between the criterion values of the ESP editing program ρ and the ideal point (the best value observed so far for each criterion during the run). A smaller scalar value corresponds to a solution that better satisfies the subproblem defined by its weight vector, and the ideal point is updated whenever a newly generated solution improves any of the criteria beyond the best value recorded so far.

The neighborhood structure of the specialized MOEA/D is defined by the Euclidean distance between weight vectors: for each subproblem, its neighborhood consists of a fixed number of subproblems whose weight vectors are closest to its own, this neighborhood size being a hyperparameter set in our implementation to one tenth of the population size. At each generation, for every subproblem a parent solution is selected from its neighborhood, the mutation operator "mutate-ops" (see Algorithm 3) is applied to produce a candidate solution, and the ideal point is updated if the candidate improves any of the criteria.

The candidate then replaces the current solution of up to a fixed number of neighboring subproblems whenever it scores better on their scalarized subproblem than their current solution, which allows a single new candidate to propagate across several related subproblems within one generation. As in the specialized NSGA-II, the hypervolume indicator is recorded after each generation, and the final Pareto-optimal set is extracted from the population using fast non-dominated sorting (see Algorithm 1, line 7) after the last generation.

III. Experimental Evaluation

A. Description of the Experiment

The Specialized NSGA-II for evolutionary programming of ESP and the described supplementary algorithms were tested on the problem of configuring computing assemblies. Each of the ESP components was characterized by its type (its functional purpose), a set of characteristics (attributes), and a cost. The compatibility of ESP components with each other was determined by the correspondence between the connector type and the ESP component type. The collected set of ESP components covered the following component types: power supply, motherboard (LGA1700 and AM5 platforms), processor, processor cooling system with a fan, graphics card with a fan, RAM modules, storage devices, case fans.

Attributes were type-specific: for the processor, the number of cores, turbo-mode frequency, L2 and L3 cache sizes, and TDP were taken into account; for the power supply, power and energy-efficiency standard were taken into account; for the RAM module, capacity, frequency, timings, and voltage were taken into account.

For each attribute, a partial order was specified, the partial order is used in (5) to numerically assess the improvement in characteristics of ESP components.

The original ESP t_{src} was represented by a budget computer configuration costing 61,900 rubles, and the target ESP t_{target} was a computer configuration costing 101,000 rubles, the target ESP could not be assembled using the available set of ESP components.

The tree of the original ESP is shown in Figure 1.

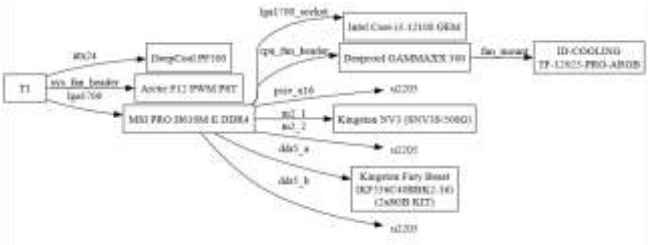

Fig. 1. Example of an ESP tree of a budgeted computer assembly

Hypervolume – a measure of the volume of the region of the criteria space dominated by the current approximation of the front relative to a fixed reference point – was used as an integral indicator of the quality of the Pareto front approximation [8]. Hypervolume remains the only widely used front quality metric that is strictly monotone with respect to the dominance relation, as detailed in [17, 18].

A larger hypervolume value corresponds to a better and more uniform approximation of the front. The hyperparameters of Algorithm 1 were chosen as follows: population size $n = 50$, number of generations $m = 200$, mutation probability $r = 0.9$. When studying the influence of the hyperparameters on the hypervolume values, we tested mutation probabilities from the set {0.1, 0.3, 0.5, 0.7, 1.0}, and population sizes from the set {10, 30, 50, 70, 100}. The hypervolume values were averaged over 20 independent runs for each parameter configuration using grid search, and the best combination of algorithm parameter values was selected based on the results. The same search procedure, with the same sets of variation probability r and population size n and the same number of independent runs, was applied to the specialized MOEA/D to obtain a directly comparable set of hypervolume statistics.

The convergence plot of the algorithm with the selected hyperparameter values is shown in Figure 2.

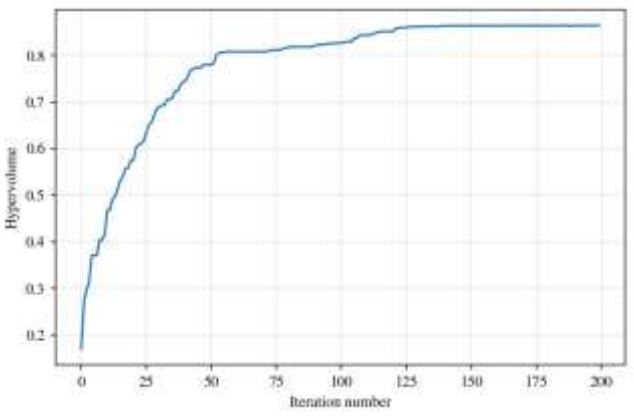

Fig. 2. Dynamics of hypervolume across generations with hyperparameters n=50, rate=0.9, averaged over 20 runs

As it is shown in Figure 2, hypervolume increases across generations and reaches a plateau, which indicates convergence of the algorithm to a stable approximation of the Pareto front. Higher values of the mutation probability provide a higher achievable level of hypervolume. The average value is about 0.658 for $r = 0.1$, about 0.757 for $r = 0.3$, about 0.792 for $r = 0.5$, about 0.822 for $r = 0.7$, and about 0.823 for $r = 1.0$. This is explained by the absence of a crossover operator in specialized NSGA-II for evolutionary programming of ESP: mutation is the only source of new ESP editing sequences.

Notably, the gain from further increasing diminishes at high probabilities: the difference between r=0.7 and $r = 1.0$ is less than 0.001, whereas the difference between $r = 0.1$ and $r = 0.3$ exceeds 0.1, indicating that the benefit of a higher mutation probability saturates once r approaches 0.7–1.0. For all studied values of r, the hypervolume increases and reaches the plateau.

The dependence of the final hypervolume on population size is monotonically increasing over the tested range: the average value rises from about 0.669 at $n = 10$ to about 0.833 at $n = 30$, about 0.841 at $n = 50$, about 0.844 at $n = 70$, and about 0.851 at $n = 100$. For small populations, the algorithm explores the space unevenly, which limits the achievable hypervolume, while the increase from $n = 10$ to $n = 30$ accounts for the largest share of the improvement (about 0.164), a further increase from $n = 30$ to $n = 50$ still yields a noticeable gain (about 0.008), whereas increasing the population size beyond $n = 50$ brings only marginal additional benefit (about 0.010 between $n = 50$ and $n = 100$).

Based on this diminishing-returns pattern and considering that the gain from further increasing the population size beyond $n = 50$ is modest relative to the added computational cost, population size was set to 50 for the main experiment.

B. Results of the Numerical Experiment

In total, in the main run the algorithm found 11 non-dominated solutions (see Table 1).

The resulting front demonstrates a trade-off between criteria: solutions with high assembly completeness and significant characteristics improvement are associated with a larger cost overrun, whereas cheap solutions have lower completeness and smaller characteristics improvement.

The algorithm forms not a single “optimal” solution, but a set of non-dominated variants, this allows the designer to choose a solution corresponding to their priorities: under a tight budget – a solution from the lower part of the front, with priority on quality – a solution from the upper part.

Application of the TOPSIS method allows ordering of these non-dominated solutions and provides the designer not with a set of equivalent alternatives but with a ranked list (the “TOPSIS Rating” column in Table 1), while retaining the possibility of choosing any other solution of the front.

Table 1. Found Pareto-optimal solutions with TOPSIS ranking

Characteristics improvement (4)	**Cost overrun (7)**	**Assembly completeness (6)**	**TOPSIS Rank (11)**
0.42	0.074	1.0	0
0.535	0.093	1.0	1
0.585	0.124	1.0	2
0.597	0.171	1.0	3
0.602	0.18	1.0	4

Characteristics improvement (4)	Cost overrun (7)	Assembly completeness (6)	TOPSIS Rank (11)
0.647	0.191	1.0	5
0.672	0.219	1.0	6
0.688	0.258	1.0	7
0.701	0.296	1.0	8
0.722	0.3	1.0	9
0.734	0.334	1.0	10
0.747	0.366	1.0	11

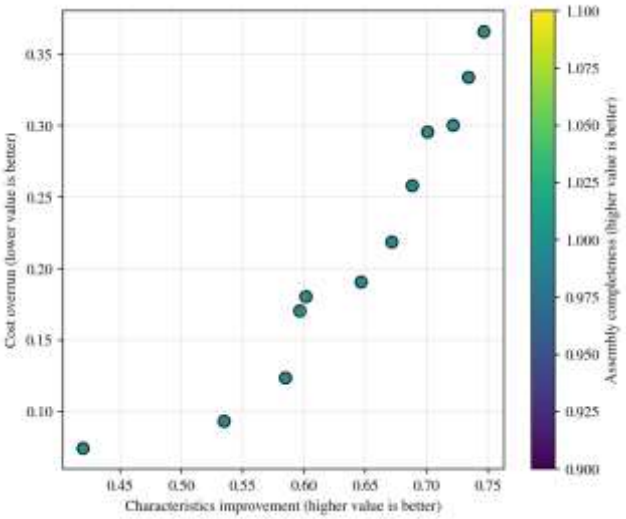

Fig. 3. Pareto front (2D) with solution numbering by rating obtained using the TOPSIS method: X axis – characteristics improvement (higher value is better), Y axis – cost overrun (lower value is better)

As it is shown in Figure 3, application of the TOPSIS method allows to achieve clear ordering of the initially incomparable Pareto-optimal solutions with respect to all three criteria (3) simultaneously.

C. Minimization of Instructions: Quantitative Assessment

The redundant instruction removal algorithm described above was applied to all 11 found ESP editing programs of the Pareto front. The results are shown in Figure 5.

As it is shown in Figure 4, instruction minimization consistently reduces the number of instructions. In particular, the number of "connect" and "disconnect" operations of ESP components, offsetting each other in the course of the evolutionary search, is significantly reduced.

On average across all 11 Pareto front solutions, the number of instructions was reduced by 32% (minimum 22%, maximum 55%), which allows to expect an effect of similar order when the algorithm is applied to ESP editing programs of other sizes. The resulting programs contain only meaningful operations that actually change the ESP structure, which makes the synthesized instructions suitable for direct application in a product composition management system.

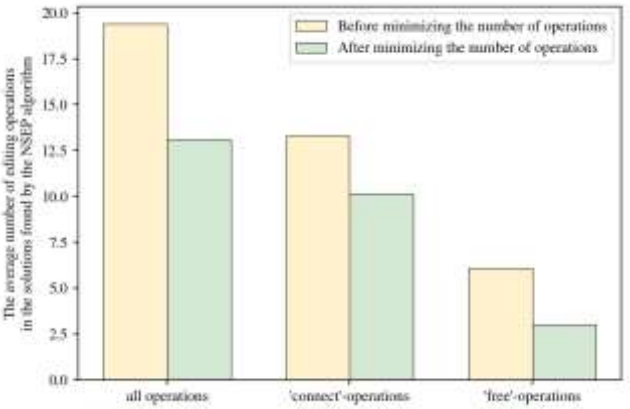

Fig. 4. Average number of ESP editing operations before and after minimization

For the solution with the highest TOPSIS rating (see rank 0 in Table 1), the ESP editing program after minimization contained only 14 operations (see Table 2). The ESP tree obtained by applying this program to the original ESP (Figure 1) is shown in Figure 6.

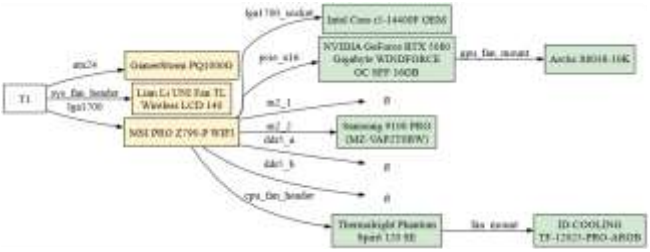

Fig. 5. Example of an ESP tree after applying a certain sequence of editing operations (modified and added nodes are highlighted in color)

Table 2. ESP editing program for the Pareto front solution with the highest TOPSIS rank 0

No.	Operation	Node	Connector	Component
1	connect	MSI PRO H610M-E DDR4	pcie_x16	NVIDIA GeForce RTX 5070 Ti MSI OC 16GB
2	free	T1	atx24	–
3	free	T1	sys_fan_header	–
4	free	T1	lga1700	–
5	connect	T1	sys_fan_header	Lian Li UNI Fan TL Wireless LCD 140
6	connect	T1	lga1700	MSI PRO Z790-P WIFI
7	connect	MSI PRO Z790-P WIFI	pcie_x16	NVIDIA GeForce RTX 5060 MSI OC 8GB
8	connect	MSI PRO Z790-P WIFI	lga1700_socket	Intel Core i9-14900KF OEM
9	connect	NVIDIA GeForce RTX 5060 MSI OC 8GB	gpu_fan_mount	Gembird D9225HM-4
10	connect	MSI PRO Z790-P WIFI	cpu_fan_header	Thermalright Phantom Spirit 120 SE
11	connect	Thermalright Phantom Spirit 120 SE	fan_mount	Thermalright TL-M12Q
12	connect	T1	atx24	GamerStorm PN1000M
13	connect	MSI PRO Z790-P WIFI	m2_2	Samsung 9100 PRO (MZ-VAP2T0BW)
14	connect	MSI PRO Z790-P WIFI	ddr5_b	Kingston Fury Beast

No.	Operation	Node	Connector	Component
				(KF556C36-BBEK2-32) (2x16GB KIT)

The ESP editing program shown in Table 2 describes exactly those changes that bring the original budget assembly (Figure 1) closer to the target configuration, which demonstrates the transition from the structure in Figure 1 to the structure in Figure 5.

D. Comparison with the Specialized MOEA/D

To evaluate the proposed specialized NSGA-II against an alternative decomposition-based approach, the specialized MOEA/D was executed on the same ESP updating task, using the same original t_{src} and target ESP t_{target}, the same set of available ESP components, and the same three criteria (4) – (7).

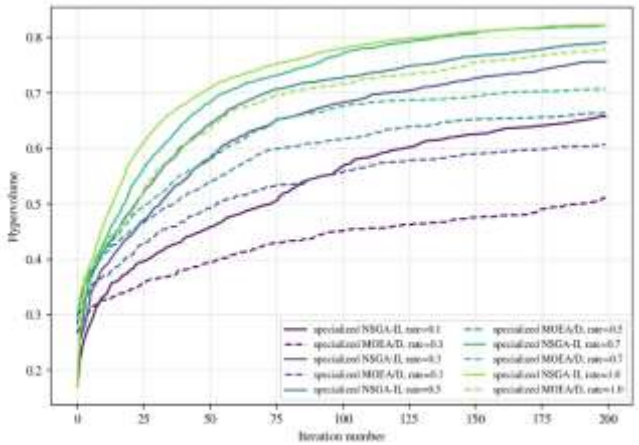

Fig. 6. Dynamics of hypervolume across generations for the specialized NSGA-II and the specialized MOEA/D at various values of the mutation/variation probability r (n=50, 200 generations, averaged over 20 runs)

As shown in Figure 6, the specialized MOEA/D converges monotonically with respect to hypervolume for all tested values of the variation probability, similarly to the specialized NSGA-II, and a higher r also yields a higher achievable hypervolume for the specialized MOEA/D. However, at every tested value of r the specialized NSGA-II reaches a higher final hypervolume than the specialized MOEA/D: the relative advantage is about 29% at $r = 0.1$ and narrows to about 6% at $r = 1.0$, where MOEA/D benefits the most from a deterministic, per-neighborhood variation.

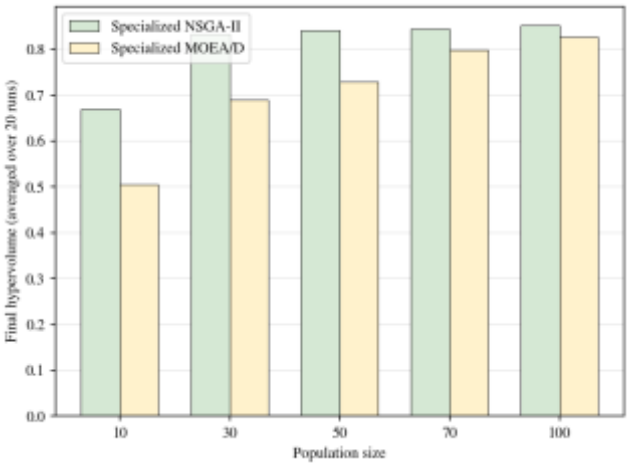

Fig. 7. Final hypervolume of the specialized NSGA-II and the specialized MOEA/D as a function of the population size n (rate=0.9, 200 generations, averaged over 20 runs)

As shown in Figure 7, the specialized NSGA-II also outperforms the specialized MOEA/D at every tested population size: the relative gain of the specialized NSGA-II over the specialized MOEA/D is about 32% at $n = 10$ and about 3% at $n = 100$.

As shown in Figure 8, the specialized MOEA/D also produces a front spanning the three criteria (4)–(7), with a similar qualitative trade-off between characteristics improvement, cost overrun, and assembly completeness as the front obtained by the specialized NSGA-II (Figure 4).

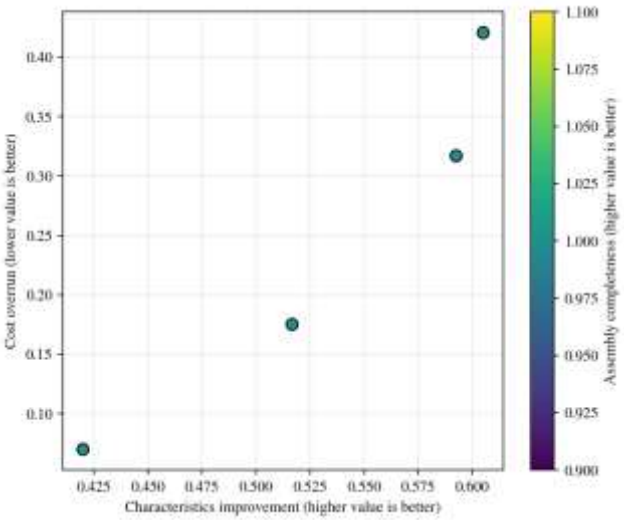

Fig. 8. Pareto front (3D) found by the specialized MOEA/D

At this configuration, the specialized NSGA-II reaches a final hypervolume of 0.847, while the specialized MOEA/D reaches 0.794 – a relative advantage of about 6.6% for the specialized NSGA-II. This confirms that, for the ESP updating problem and the chosen population size, the specialized NSGA-II provides a moderately better approximation of the Pareto front than the specialized MOEA/D.

IV. CONCLUSION

The article proposes a novel specialized NSGA-II for evolutionary programming of electronic structure of a product for solving the problem of multi-criteria updating of the electronic structure of a product.

The ESP is represented as a tree with an explicit connector model (1), which allows us to correctly describe the compatibility of ESP components and structural operations on the product. Three optimization criteria are introduced – characteristics improvement, assembly completeness, and cost overrun (4) – (7), the values of these criteria belong to the range [0, 1]. These criteria cover the main engineering requirements for the product configuration task; the exponential complexity of exhaustive search over ESP editing programs is shown, justifying the use of evolutionary methods.

The proposed algorithm is a specialized version of NSGA-II, but is based on evolutionary programming over ESP trees. There is no crossover operator in the algorithm, so mutation adds or removes ESP editing operations, and post-processing removes redundant instructions that can be generated during the process of evolutionary search. The algorithm operates directly on the structure of the ESP tree, without encoding solutions as numerical parameter vectors, this also distinguishes it from classical implementations of NSGA-II.

The specialized NSGA-II for evolutionary programming of electronic structure of a product algorithm forms a set of non-dominated ESP configurations with a trade-off between quality, completeness and cost. Application of the TOPSIS method with entropy weights allows ordering Pareto-optimal solutions for presentation to the designer in decreasing order of preference (11), and the removal of redundant instructions reduces the sizes of ESP editing programs of the Pareto front by an average of 32%.

To validate the choice of NSGA-II as the underlying multi-criteria search mechanism, a specialized version of the decomposition-based MOEA/D algorithm was additionally developed on the same ESP tree representation and mutation operator. The comparison showed that the specialized NSGA-II consistently achieves a higher hypervolume value than the specialized MOEA/D across all tested population sizes and variation probabilities, with the relative advantage ranging from about 32% for small populations ($n = 10$) to about 3% for large populations ($n = 100$), and about 6.6% at the population size and generation count used to construct the reported Pareto front. This result supports the choice of the non-dominated sorting-based approach for the ESP updating problem at moderate population sizes, while also indicating that the decomposition-based specialized MOEA/D remains a viable and computationally lighter alternative when larger populations can be afforded.

The proposed approach automates the routine task of configuring products when replacing ESP components: instead of manual scanning through available options, the designer receives an ordered list of specific ESP editing operations with numerical scores for each criterion. The method is applicable in PLM/PDM systems to support decision-making when updating the product composition, as well as when planning maintenance and replacement of ESP components in accordance with configuration management procedures [19] and the procedures for making changes to electronic design documentation [20].

REFERENCES

[1] GOST R 2.053-2023. Unified system for design documentation. Electronic structure of a product. Basic principles. — Moscow: Federal Agency on Technical Regulating; Metrology; Russian Institute for Standardization, 2023.

[2] GOST 2.054-2013. Unified system for design documentation. Electronic description of a product. Basic principles. — Moscow: Interstate Council for Standardization, Metrology; Certification; Standartinform, 2013.

[3] GOST R 2.525-2024. Unified system for design documentation. Constructive electronic structure of a product. Data format. — Moscow: Federal Agency on Technical Regulating; Metrology; Russian Institute for Standardization, 2024.

[4] GOST R 56136-2014. Life cycle management of military products. Terms and definitions. — Moscow: Federal Agency on Technical Regulating; Metrology; Standartinform, 2016.

[5] GOST R 15.000-2016. System of product development and launching into manufacture. Basic principles. — Moscow: Federal Agency on Technical Regulating; Metrology; Standartinform, 2019.

[6] Gabrielyan G. A. Electronic product structure synthesis based on language models, constraint programming, and intermediate representation translators // Electronic scientific journal "IT-Standard". — 2025. — Is. 4. — P. 124–137.

[7] Andrianova E. G., Gabrielyan G. A. A method of updating electronic product structures based on an event-oriented approach using large language models // Electronic scientific journal "IT-Standard". — 2026. — Is. 2. — P. 120–131.

[8] Zitzler E., Thiele L. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach // IEEE Transactions on Evolutionary Computation. — IEEE, 1999. — Vol. 3, No. 4 — P. 257–271.

[9] Zolotarev M. A. Methods of multi-criteria optimization of technological objects: a systematic review of scientific publications for the period 2013–2023 // Vestnik of Samara State Technical University. Technical Sciences Series. — 2024. — Is. 2. — P. 25-47.

[10] Fogel D. B. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. — Piscataway, NJ: IEEE Press, 1995.

[11] Koza J. R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. — Cambridge, MA: MIT Press, 1992.

[12] Deb K., Pratap A., Agarwal S., Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II // IEEE Transactions on Evolutionary Computation. — IEEE, 2002. — Vol. 6, No. 2 — P. 182–197.

[13] Garagulova A. K., Gorbacheva D. O., Chirkov D. V. Comparative analysis of MOGA and NSGA-II on the case study of optimization for the profile of the hydraulic turbine runner // Computational Technologies. — 2018. — Vol. 23, No. 5 — P. 21–36.

[14] Subtil R. F., Carrano E. G., Souza M. J., Takahashi R. H. Using an enhanced integer NSGA-II for solving the multiobjective generalized assignment problem / IEEE congress on evolutionary computation. — IEEE, 2010. — P. 1–7.

[15] Hwang C.-L., Yoon K. Multiple Attribute Decision Making: Methods and Applications. — Berlin: Springer, 1981. — T. 186.

[16] Zhang Q., Li H. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition // IEEE Transactions on Evolutionary Computation. — IEEE, 2007. — Vol. 11, No. 6 — P. 712–731.

[17] Shang K., Ishibuchi H., He L., Pang L. M. A Survey on the Hypervolume Indicator in Evolutionary Multiobjective Optimization // IEEE Transactions on Evolutionary Computation. — IEEE, 2021. — T. 25, V. 1. — P. 1–20.

[18] Guerreiro A. P., Fonseca C. M., Paquete L. The Hypervolume Indicator: Computational Problems and Algorithms // ACM Computing Surveys. — ACM, 2021. — T. 54, V. 6. — P. 1–42.

[19] GOST R ISO 10007-2019. Quality management. Guidelines for configuration management. — Moscow: Federal Agency on Technical Regulating; Metrology; Standartinform, 2019.

[20] GOST R 2.504-2021. Unified system for design documentation. Electronic design documentation. Rules for making changes. — Moscow: Federal Agency on Technical Regulating; Metrology; Russian Institute for Standardization, 2021.

Gayk A. Gabrielyan, Lecturer, Department of Corporate Information Systems, Institute of Information Technologies, MIREA – Russian Technological University (78, Vernadsky pr., Moscow, 119454). gabrielyan@mirea.ru. https://orcid.org/0009-0009-1033-776X