УДК 004.4 https://doi.org/10.15407/pp2025.03.039

*В.І. Шинкаренко, О.В. Макаров*

# КОНСТРУКТИВНО-ПРОДУКЦІЙНЕ МОДЕЛЮВАННЯ ХРОМОСОМ ГЕНЕТИЧНОГО АЛГОРИТМУ ІЗ ЗАКОДОВАНИМИ АЛГОРИТМАМИ СОРТУВАННЯ

У разі структурної адаптації алгоритмів із використанням генетичного алгоритму однією із проблем є кодування структури алгоритмів у хромосому. У статті розглядається підхід до структурної адаптації та формування ефективних алгоритмів сортування на основі парадигми конструктивно-продукційного моделювання. Запропоновано метод представлення хромосом у вигляді закодованих структур, що відображають різні варіанти алгоритмів сортування. Такий підхід дозволяє формувати простір пошуку рішень не лише у вигляді набору числових параметрів, а й у вигляді цілісних програмних конструкцій. Описано операції підстановки, часткового виведення та композиції, які забезпечують синтез нових алгоритмів сортування. Для формування і відбору функціонально еквівалентних алгоритмів застосовано генетичний алгоритм. Наведено приклади побудови хромосом-дерев, що кодують сортувальні процедури різної складності. Реалізовано програму для генерації та еволюційного вдосконалення хромосом із закодованими алгоритмами сортування. продемонстровано застосування конструктивно-продукційного моделювання для вирішення задачі кодування структурно різних, але функціонально еквівалентних алгоритмів у хромосомах. Результати експериментів підтвердили, що застосування конструктивно-продукційного моделювання забезпечує підвищення різноманітності популяцій та прискорює знаходження ефективних рішень у порівнянні з класичними генетичними алгоритмами. Запропонована методика може бути використана для автоматизованого конструювання алгоритмів, оптимізації їхньої структури та адаптації до специфічних умов використання.

Ключові слова: конструктивно-продукційне моделювання, алгоритми сортування, часова ефективність, програмне забезпечення, інформаційні технології, генетичний алгоритм.

*V.I. Shinkarenko, O.V. Makarov*

# CONSTRUCTIVE-SYNTHESIZING MODELING OF THE GENETIC ALGORITHM CHROMOSOMES WITH ENCODED SORTING ALGORITHMS

In the structural adaptation of algorithms using a genetic algorithm, one of the challenges is encoding the structure of algorithms into a chromosome. This article explores an approach to structural adaptation and the development of efficient sorting algorithms based on the paradigm of constructive-synthesizing modeling. A method is proposed for representing chromosomes as encoded structures corresponding to various variants of sorting algorithms. This approach allows the solution search space to be formed not only as a set of numerical parameters but also as complete software. Operations of substitution, partial inference, and composition are described, which enable the synthesis of new sorting algorithms. A genetic algorithm is applied to generate and select functionally equivalent algorithms. Examples are provided of constructing chromosome trees that encode sorting procedures of varying complexity. A program has been implemented for generating and evolutionarily improving chromosomes with encoded sorting algorithms. The application of constructive-synthesizing modeling to the problem of encoding structurally different but functionally equivalent algorithms into chromosomes is demonstrated. Experimental results confirmed that usage of constructive-synthesizing modeling increases population diversity and accelerates the discovery of efficient solutions compared to classical genetic algorithms. The proposed methodology can be used for automated algorithm construction, optimization of their structure, and adaptation to specific usage conditions.

Key words: constructive-synthesizing modeling, sorting algorithm, time efficiency, software, information technology, genetic algorithm.

## Вступ

Структурна адаптація алгоритмів за показниками часової ефективності передбачає зміну складових цих алгоритмів. Для формування і відбору функціонально еквівалентних алгоритмів застосовується генетичний алгоритм. Тут постає проблема кодування алгоритмів, що підлягають адаптації, у вигляді хромосоми. Це ускладнюється ще й можливістю розподілу обчислень у різних частинах даних. Еталонними алгоритмами у програмуванні є алгоритми сортування. Тому на прикладі алгоритмів сортування у цій роботі продемонстровано застосування конструктивно-продукційного моделювання для вирішення задачі кодування структурно різних, але функціонально еквівалентних алгоритмів у хромосомах.

Незважаючи на наявність великої кількості фундаментальних алгоритмів сортування, таких як Insertion Sort, Quick Sort [1] або Merge Sort [2], які широко використовуються завдяки своїй простоті реалізації та зрозумілій логіці, проблема підвищення їхньої ефективності залишається актуальною. Зростання обсягів даних, розвиток багатоядерних і паралельних архітектур, а також потреба в оптимізації під кеш-пам'ять і специфічні структури даних зумовлюють появу нових підходів. Саме тому активно ведуться наукові дослідження, спрямовані як на удосконалення класичних методів (Insertion Sort, Merge Sort, Quicksort), так і на створення нових алгоритмів, здатних поєднувати простоту, стабільність і високу продуктивність у сучасних умовах.

До найбільш відомих прикладів побудови алгоритмів на основі кількох класичних належить Timsort [3] – гібридний алгоритм, що поєднує переваги Merge Sort та Insertion Sort і використовується в стандартних бібліотеках Python та Java, а також Introsort [4], який адаптивно перемикається між Quick Sort, Heap Sort і Insertion Sort залежно від глибини рекурсії, забезпечуючи гарантовану ефективність у найгіршому випадку.

Попри наявність ефективних рішень, розробка нових алгоритмів сортування продовжується. Одним із сучасних підходів до вдосконалення алгоритмів сортування є розробка паралельних методів, що враховують особливості розподілу даних та ефективність операцій ранжування. Наприклад, у роботі [5] запропоновано модель паралельного алгоритму сортування з формуванням рангу, яка демонструє переваги у швидкодії та масштабованості для великих обсягів даних.

Сучасні дослідження пропонують вдосконалення класичних алгоритмів сортування. Наприклад, Multi-Pivot Quicksort [6] – використання декількох опорних елементів (k-pivot), що покращує ефективність роботи із кешем та знижує кількість порівнянь. Однією із можливих реалізацій є варіант із трьома опорними елементами, який продемонстрував приріст продуктивності у 7-8% порівняно із класичним алгоритмом з 1 опорним елементом.

На відміну від класичного сортування вставками, у [7] масив має дві відсортовані послідовності на початку і в кінці масиву, а невідсортована частина знаходиться між ними. Також важливою перевагою є можливість вставляти більше одного елемента на правильні позиції за одну ітерацію. Експериментально алгоритм показав менший середній час виконання, ніж Quick Sort для відносно невеликих масивів обсягом до 1500 елементів.

Adaptive ShiversSort [8] – алгоритм сортування, заснований на TimSort. Особливо ефективний для сортування частково впорядкованих даних. Основна ідея полягає в тому, що масив спочатку розбивається на вже відсортовані підпослідовності, які виявляються під час одного проходу. Adaptive ShiversSort є 3-aware алгоритмом – тобто, вибираючи операції злиття, він аналізує лише три верхні елементи стека послідовностей, що знижує складність керування процесом. Алгоритм додатково приймає цілочислений параметр, який змінює логіку злиття послідовностей.

Magnetic Bubble Sort [9]: Іноваційний метод, який замість поодиноких перестановок сусідніх елементів, оперує «блоком» елементів. Такі блоки поводяться

як "магнітики", притягуючись або відштовхуючись залежно від значень. Такий підхід дозволяє «переносити» відразу кілька елементів, що економить кількість проходів масивом у випадках, коли в масиві багато повторів. У деяких тестах час сортування зменшився на 20 % порівняно з bubble sort.

One-By-One (OBO): A Fast Sorting Algorithm [10]: Новий in-place алгоритм сортування. Ідея полягає у поступовому "вивільненні" кожного елемента на правильне місце по черзі (one-by-one), комбінуючи стратегічне локальне впорядкування і глобальну оптимізацію. Такий підхід особливо ефективний у разі невеликих масивів і масивів, що частково або майже впорядковані, оскільки він зменшує непотрібні переміщення шляхом прямого позиціонування. OBO демонструє кращий середній час виконання в порівнянні з класичними квадратичними алгоритмами, наприклад Selection Sort і Bubble Sort.

Для відбору кращого за часовими показниками алгоритму сортування доцільне використання генетичного алгоритму [11] (ГА), що є поширеним інструментом для розв'язання складних оптимізаційних задач, які виникають у різних галузях науки і техніки. Він базується на принципах природного відбору та генетики і використовує популяцію можливих розв'язків, які еволюціонують з покоління в покоління [12]. Одним із ключових аспектів генетичного алгоритму є формування хромосом, які представляють можливі розв'язки задачі, у даному випадку конкретні алгоритми сортування.

Для формування хромосом у цій роботі використано конструктивно-продукційне моделювання (КПМ) – підхід, який дозволяє створювати складні структури шляхом застосування продукційних правил [13]. Цей метод широко використовується в різних галузях, таких як комп'ютерна графіка, моделювання біологічних систем та автоматизоване проєктування. Однією з основних переваг використання КПМ у генетичних алгоритмах є можливість створення структур, які відповідають певним обмеженням та вимогам задачі. Це дозволяє зменшити розмір пошукового простору та підвищити ефективність алгоритму. Крім того, КПМ дозволяє легко інтегрувати доменні знання у процес генерації хромосом, що може значно покращити якість розв'язків.

У цій статті буде детально розглянуто процес формування хромосом за допомогою КПМ, включаючи опис правил продукції.

## Конструктивно-продукційна модель формування хромосоми

Узагальненим конструктором [13] називається трійка

$$C = \langle M, \Sigma, \Lambda \rangle, \qquad (1)$$

де $M$ – неоднорідний розширюваний носій; $\Sigma$ – сигнатура операцій та відношень; $\Lambda$ – інформаційне забезпечення конструювання (ІЗК): онтологія, мета, правила та обмеження, початкова форма та умови закінчення.

Визначимо спеціалізацію конструктору $C_{SA}$ – моделі формування хромосоми, що кодує алгоритм сортування. Виконаємо операцію уточнюючого перетворення узагальненого конструктора $C$.

$$C = \langle M, \Sigma, \Lambda \rangle \quad {}_{s}\!\longmapsto C_{SA} = \langle M_{SA}, \Sigma_{SA}, \Lambda_{SA} \rangle, \qquad (2)$$

де $M_{SA}$ – носій, що включає, зокрема, термінальний ($T_1$) і нетермінальний($N_1$) алфавіт, а також множину правил продукції $\Psi$ з правилами $\psi_i = \langle s_i, g_i \rangle \in \Psi$, де i – номер правила, $s_i$ – послідовність відношень підстановки, $g_i$ – послідовність операцій над атрибутами. $\Sigma_{SA}$ – операції та відносини на елементах $M_{SA}$. ІЗК $\Lambda_{SA} \supset \Lambda$.

Для формування конструкцій у вигляді хромосоми у $\Sigma_{SA}$ передбачені операції підстановки, часткового та повного виводу. Вивід виконується від початкового нетерміналу, який є початковою формою ітераційним виконанням операцій часткового виводу. Операція часткового виводу обирає одне з правил із множини $\Psi$ та виконує заміну лівої частини правила на праву у поточній формі та виконує операції над атрибутами. Вона використовує операцію підстановки: саме зміна поточної форми і є частиною операції повного виводу, починаючи з початкового нетерміналу і закінчуючи готовою конструкцією.

У цій роботі $g_i = \langle g_{i,1}, g_{i,2} \rangle \in \Psi$. Операції виконуються у наступній послідовності: спочатку виконуються операції над атрибутами $g_{i,1}$ потім операції підстановки $s_i$ і в кінці – $g_{i,2}$.

ІЗК $\Lambda_{SA}$ містить визначення, доповнення та обмеження, які уточнюють алфавіт, атрибути носія, відносини підстановки задають особливості виконання операцій підстановки та виведення.

Нетермінальний алфавіт $N_1 = \{{}_{\phi}\alpha_i\}$ складається з допоміжних символів з атрибутами, що позначаються грецькими літерами.

Термінальним алфавітом $T_1$ є множина генів, які утворюють хромосому, відповідну алгоритму сортування. Гени відповідають частинам відомих алгоритмів сортування, алгоритмам передобробки [14,15] даних і допоміжним функціям. Наведемо список усіх терміналів – генів, відповідно до:

- BSF (Bubble sort forward) – одного проходу алгоритму сортування бульбашкою;
- BSB (Bubble sort backward) – одного проходу алгоритму сортування бульбашкою у зворотному порядку;
- FSB (Find swap biggest element) – пошуку найбільшого елементу у невідсортованій частині масиву із перестановкою на поточне місце;
- FSS (Find swap smallest element) – пошуку найменшого елементу у невідсортованій частині масиву із перестановкою на поточне місце;
- SEEI (Single element end insertion) – операції вставки поточного елемента у відсортовану частину в кінці масиву;
- SEI (Single element insertion) – операції вставки поточного елемента у відсортовану частину на початку масиву;
- SIS (Split in subarrays) – операції розділення невідсортованої частини масиву на дві рівні частини для подальшого сортування кожної з них окремо. Також зберігання покажчиків та розмірів масивів для подальшого виконання операції злиття;
- P (Partition) – встановлення елемента під індексом і на правильне місце у відсортованому масиві і перекидання менших елементів ліворуч, а більших – праворуч;
- FS (Final sort) – одного із загальновідомих алгоритмів сортування (quicksort, mergesort, heapsort);
- PUA (Pop unsorted array) – дістання покажчика і розміру наступної невідсортованої частини для сортування;
- ESS (End subarray sort) – допоміжної операції, яка закриває фігурні дужки відкриті попередніми операціями для перевірки індикатора відсортованості;
- ML (Merge left) – збереження покажчиків на відсортовану частину на початку та решту масиву для подальшого злиття в один відсортований масив;
- MR (Merge right) – збереження покажчиків на відсортовану частину в кінці та решту масиву для подальшого злиття в один відсортований масив;
- QP (Quick Preprocess) – передбачення позиції елемента у відсортованому масиві й перестановка;
- PM (Preprocess with Memory) – те саме, що і QP, але передбачені індекси запам'ятовуються і перестановка виконується на наступне раніше не передбачувану позицію;
- SR (Sort Ranges) – розвертання усіх послідовностей елементів, що йдуть у зворотному порядку;
- BLQP (Block Local Quick Preprocessing) – розбивка сортованих даних на блоки певного обсягу і виконання швидкої перестановки у межах блоку;
- BLPM (Block Local Preprocessing with Memory) – розбивка сортованих даних на блоки певного обсягу і виконання перестановки із пам'яттю у межах блока;
- BGQP (Block Global Quick Preprocessing) – визначення позиції елемента (як у швидкій передобробці) і виконання перестановки тільки

якщо передбачена позиція перебуває у межах блоку;

- BGPM (Block Global Preprocessing with Memory) – визначення позиції елемента (як у передобробці з пам'яттю) і виконання перестановки, тільки якщо передбачена позиція є у межах блоку.

У процесі формування хромосоми [16] операції підстановки (на основі відпо-

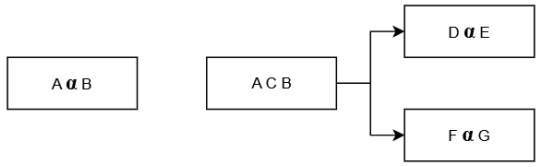


Рис 1. Початкова форма (зліва) і форма після виконання операції підстановки (справа)

відних відношень підстановки) реалізують додавання терміналів (генів) до вузлів дерева і нових вузлів до дерева. Розглянемо відношення підстановки.

Відношення $\beta \ _{\tau}\!\rightarrow \gamma$ дає можливість виконати заміну $\beta$ на $\gamma$ у збудованій частині хромосоми, якщо в ній є $\beta$ і атрибут доступності $\tau$ дорівнює true.

$\beta \ _{\tau}\!\rightarrow \gamma \begin{matrix}\nearrow \rho \\ \searrow \delta\end{matrix}$, де $\beta$, $\gamma$, $\delta$ і $\rho$ –деякі послідовності терміналів і нетерміналів. Відношення дає можливість виконати заміну $\beta$ на $\gamma$ у збудованій частині хромосоми, а конкретно у поточному вузлі хромосоми-дерева, якщо в ній є $\beta$ і атрибут доступності $\tau$ дорівнює true. Відношення «↗» і «↘» дають можливість додати лівий і правий вузли нащадки до поточного вузла дерева і додати $\rho$ і $\delta$ до відповідних вузлів.

Розглянемо наступний приклад правила підстановки.

$$\alpha \ _{\tau}\!\rightarrow C \begin{matrix}\nearrow D \cdot \alpha \cdot E \\ \searrow F \cdot \alpha \cdot G\end{matrix}, \qquad (3)$$

Поточна послідовність $A \cdot \alpha \cdot B$, де $\alpha$ – нетермінал і $A, B$ – термінали. У ході виконання операції підстановки на основі відношення підстановки (3) буде додано ген C до початкової послідовності генів поточного вузла (Node level 1), два вузли-нащадки до поточного вузла. Також додано ген D і нетермінал $\alpha$ до початкової послідовності та ген E до кінцевої послідовності лівого вузла нащадка. Ген F і нетермінал $\alpha$ буде додано до початкової послідовності правого вузла нащадка, а ген G – відповідно до кінцевої. Результат підстановки представлено на рис. 1. Далі підстановки будуть продовжені відповідно для нетерміналів лівого та правого вузлів.

Визначені такі операції над атрибутами: :=(a, b) – присвоєння значення b змінній a; +(a, b, c) – додавання аргументів b і c, результат зберігається у a; ==(a, b, c) – якщо b дорівнює c, зберігає true у a, інакше – false; ∧(a, b, c) – виконання операції логічне «і» над агрументами b і c, результат зберігається у a; >(a, b, c) – порівнює аргументи b і c. Якщо b більше за c, присвоює a значення true, інакше – false; <(a, b, c) – порівнює аргументи b і c. Якщо b менше за c, присвоює a значення true, інакше – false; >=(a, b, c) – порівнює аргументи b і c. Якщо b більше або рівне c, присвоює a значення true, інакше – false; <=(a, b, c) – порівнює аргументи b і c. Якщо b менше або рівне c, присвоює a значення true, інакше – false.

Атрибути терміналів і нетерміналів поточної форми:

- $b_1/b_2$ – ознаки того, що елементи у відсортованій частині на початку/кінці масиву менші/більші за усі інші;
- cD – поточна глибина дерева-хромосоми;
- nG – кількість генів у поточному вузлі дерева-хромосоми;
- pGU – ознака того, що попередній використаний ген є будь-яким геном передобробки;
- MD – значення максимальної глибини дерева-хромосоми;
- MGBS – значення мінімальної кількості генів у вузлі, після досягнення якого стають доступними гени розбиття.

Початкові умови конструювання: $\alpha$ – нетермінал, з якого починається виведення.

Умови завершення конструювання: Хромосома повністю сконструйована (поточна форма не містить нетерміналів).

## Правила підстановки конструктора формування хромосом

У підході, що розглядається, після додавання кожного нового вузла до дерева-хромосоми, одразу додаються ген PUA до початкової послідовності генів і ген ESS – до кінцевої.

Перша група правил містить відношення підстановки генів, що відповідають частинам існуючих алгоритмів сортування. Від значення атрибута залежить кількість доданих терміналів і значення атрибутів, які змінюються після виконання підстановки.

Детально розглянемо правило $\psi_1 = \langle s_1, \langle g_{1,1}, g_{1,2} \rangle\rangle$.

До операцій підстановки виконуються операції над атрибутами $g_{1,1}$. Встановлюються флаги $\tau_1$ і $\tau_2$, що роблять доступними відношення $s_1$.

Виконання операції підстановки за вибраним відношенням підстановки $\alpha \ {}_{\tau_1}\!\!\rightarrow FSS \cdot \alpha$ забезпечить додавання терміналу FSS до початкової послідовності генів поточного вузла і нетермінала $\alpha$. Якщо значення атрибута $\tau_2$ буде дорівнювати true, то доступним стає відношення $\alpha \ {}_{\tau_2}\!\!\rightarrow ML \cdot FSS \cdot \alpha$ і замість одного буде додано два термінали: ML і FSS.

Після операцій підстановки виконуються операції над атрибутами $g_{1,2}$. Поточні значення деяких атрибутів можуть змінювати цю поведінку. Якщо значення атрибута $b_1$ дорівнює true, то значення атрибута nG збільшиться на 1, і на 2 якщо $b_1$ дорівнює false. Далі втановлюються значення атрибутів pGU = false і $b_1$ = true.

$$s_1 = \langle {}^{\alpha\ \tau_1 \rightarrow FSS\cdot\alpha,}_{\alpha\ \tau_2 \rightarrow ML\cdot FSS\cdot\alpha.} \rangle;$$
$$g_{1,1} = \langle {}^{==(\tau_1, b_1, \text{true}),}_{==(\tau_2, b_1, \text{false}).} \rangle;$$
$$g_{1,2} = \langle {}^{\text{if}\left(b_1, +(nG, nG, 1), +(nG, nG, 2)\right),}_{:=(pGU, \text{false}),} {}_{:=(b_1, \text{true}).} \rangle \quad (4)$$

Аналогічно до правила $\psi_1$ у правилі $\psi_2 = \langle s_2, \langle g_{2,1}, g_{2,2} \rangle\rangle$ до початкової послідовності генів додається один (FSB) або два ($MR \cdot FSB$) термінали залежно від значення атрибутів $\tau_1$ і $\tau_2$.

$$s_2 = \langle {}^{\alpha\ \tau_1 \rightarrow FSB\cdot\alpha,}_{\alpha\ \tau_2 \rightarrow MR\cdot FSB\cdot\alpha.} \rangle;$$
$$g_{2,1} = \langle {}^{==(\tau_1, b_2, \text{true}),}_{==(\tau_2, b_2, \text{false}).} \rangle;$$
$$g_{2,2} = \langle {}^{\text{if}\left(b_2, +(nG, nG, 1), +(nG, nG, 2)\right),}_{:=(pGU, \text{false}),} {}_{:=(b_1, \text{true}).} \rangle \quad (5)$$

Наступні правила містять відношення підстановки генів SEI і SEEI:

$$s_3 = \langle \alpha \rightarrow SEI \cdot \alpha \rangle;\ g_{3,1} = \langle \varepsilon \rangle;$$
$$g_{3,2} = \langle {}^{+(nG, nG, 1),}_{:=(pGU, \text{false}),} {}_{:=(b_1, \text{false}).} \rangle \quad (6)$$

$$s_4 = \langle \alpha \rightarrow SEEI \cdot \alpha \rangle;\ g_{4,1} = \langle \varepsilon \rangle;$$
$$g_{4,2} = \langle {}^{+(nG, nG, 1),}_{:=(pGU, \text{false}),} {}_{:=(b_2, \text{false}).} \rangle \quad (7)$$

Символ ε – порожньо, означає відсутність операцій над атрибутами до операцій підстановки.

Після виконання підстановки значення атрибута nG збільшується на 1, pGU стає рівним false. Відповідно для першого і другого правила виконується b1 = false або b2 = false.

Правило $\psi_5 = \langle s_5, \langle g_{5,1}, g_{5,2} \rangle\rangle$ містить відношення підстановки гена BSF:

Якщо значення атрибута $\tau_1$ істинне, доступне відношення ${}_{\tau_1}\!\!\rightarrow$, і до поточного вузла буде додано ген BSF і нетермінал $\alpha$, для якого будуть продовжені підстановки. Якщо істинне значення $\tau_2$, то ${}_{\tau_2}\!\!\rightarrow$ доступне. Буде додано два гени MR і BSF та нетермінал $\alpha$. Правило підстановки буде виконуватись, якщо хоча б одне відношення доступне.

$$s_5 = \langle {}^{\alpha\ \tau_1 \rightarrow BSF\cdot\alpha,}_{\alpha\ \tau_2 \rightarrow MR\cdot BSF\cdot\alpha.} \rangle;\ g_{5,1} = g_{2,1};$$
$$g_{5,2} = g_{2,2} \quad (8)$$

Правило $\psi_6 = \langle s_6, \langle g_{6,1}, g_{6,2} \rangle\rangle$ містить відношення підстановки для додавання (під час реалізації конструктора) гена BSB у частково сформовану хромосому:

$$s_6 = \langle {}^{\alpha\ \tau_1 \rightarrow BSB\cdot\alpha,}_{\alpha\ \tau_2 \rightarrow ML\cdot BSB\cdot\alpha.} \rangle;\ g_{6,1} = g_{1,1};$$
$$g_{6,2} = g_{1,2} \quad (9)$$

Наступні два правила містять відношення підстановки генів розбиття P і SIS – у процесі виконання підстановки, за

якими до поточного вузла додаються два вузли-нащадки. Виконання підстановок завершується для поточного вузла і буде продовжено для вузлів-нащадків.

У результаті підстановки $s_7$ до поточного вузла додається ген P і два вузли-нащадки (лівий і правий). Далі паралельно виконуються 2 підстановки. Для доданих нетерміналів будуть продовжені підстановки відповідно у лівий і правий вузли. Гени, що стоять зліва від нетермінала $\alpha$, будуть додані у початкову послідовність генів вузла, справа – у кінцеву послідовність.

$$\begin{gathered} s_7 = \langle \alpha \ {}_{\tau_1}\!\rightarrow P^{\nearrow PUA\cdot\alpha\cdot ESS}_{\searrow PUA\cdot\alpha\cdot ESS}\rangle; \\ g_{7,1} = \langle {}^{<(f_1,\mathrm{cD},\mathrm{MD}),}_{>=(f_2,\mathrm{nG},\mathrm{MGBS}),}{}^{\wedge(\tau_1,f_1,f_2).}\rangle; \\ g_{7,2} = \langle {}^{+(\mathrm{cD},\mathrm{cD},1),}_{:=(\mathrm{pGU},\mathrm{false}),}{}^{:=(\mathrm{nG},0).}\rangle \end{gathered} \quad (10)$$

Правило $\psi_8 = \langle s_8, \langle g_{8,1}, g_{8,2}\rangle\rangle$ містить відношення підстановки гена SIS:

$$\begin{gathered} s_8 = \langle \alpha \ {}_{\tau_1}\!\rightarrow SIS^{\nearrow PUA\cdot\alpha\cdot ESS}_{\searrow PUA\cdot\alpha\cdot ESS}\rangle, \\ g_{8,1} = g_{7,1};\ g_{8,2} = g_{7,2} \end{gathered} \quad (11)$$

Правило $\psi_9 = \langle s_9, \langle g_{9,1}, g_{9,2}\rangle\rangle$ містить відношення підстановки гена фінального сортування FS. Даний ген використовується для гарантованого відсортування поточної частини масиву сортованих даних, тому в результаті підстановки до послідовності не додаються нетермінали і після не виконуються операції над атрибутами.

Наступне правило $\psi_{10} = \langle s_{10}, \langle g_{10,1}, g_{10,2}\rangle\rangle$ містить відношення підстановки генів передобробки. До операції підстановки виконується операція над атрибутом і встановлюється значення $\tau_1$. У процесі виконання операції підстановки до послідовності терміналів додається ген передобробки і нетермінал $\alpha$. Символ "|" позначає відношення «або»: можливість виконання лише однієї із зазначених підстановок. Потім значення атрибута nG збільшується на 1, pGU стає рівним false.

$$\begin{gathered} s_{10} = \langle \alpha \ {}_{\tau_1}\!\rightarrow QP\cdot\alpha \mid PM\cdot \\ \alpha \mid SR\cdot\alpha \mid BLQP\cdot\alpha \mid BLPM\cdot \\ \alpha \mid BGQP\cdot\alpha \mid BGPM\cdot\alpha\rangle; \\ g_{10,1} = \langle == (\tau_1, \mathrm{pGU}, \mathrm{false})\rangle; \\ g_{10,2} = \langle {}^{+(\mathrm{nG},\mathrm{nG},1),}_{:=(\mathrm{pGU},\mathrm{true}).}\rangle \end{gathered} \quad (12)$$

## Розширений приклад формування хромосоми

Розглянемо приклад покрокового процесу формування хромосоми-дерева. На початку встановлюються наступні початкові значення параметрів конструювання і атрибутів: MD = 3, MGBS = 1, b1 = true, b2 = true, cD = 1, nG = 0, pGU = false.

Розглянемо послідовність підстановок для кореневого вузла хромосоми рівня 1 під номером 1. Першим обирається правило підстановки $\psi_{10}$, додається ген швидкої передобробки QP, встановлюються значення атрибутів $nG = 1; pGU = true;$. Наступними обираються два однакових правила $\psi_5$. Додаються гени BSF, встановлюються атрибути $nG = 3; pGU = false;$. Останнім обирається правило $\psi_8$, що завершує вибір правил для поточного вузла, додає ген SIS і встановлює значення атрибутів cD=1; pGU=false; nG=0;. Також додає два вузли - нащадки і відповідні термінали і нетермінали у їхній пос-

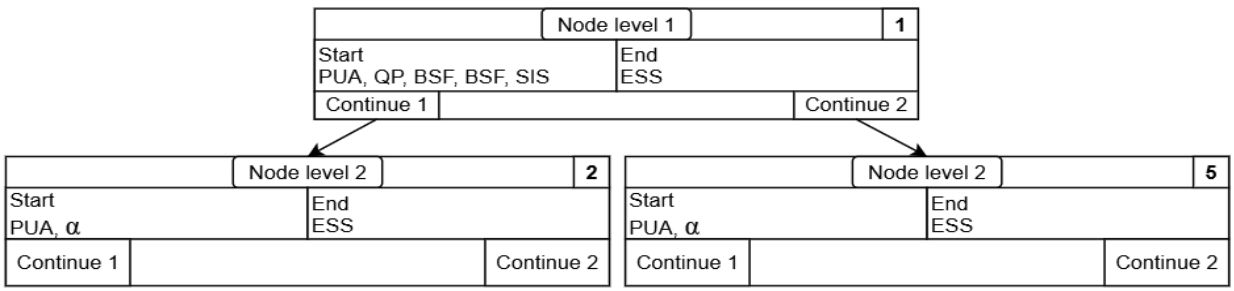


Рис. 2. Змінена форма після застосування правил $\boldsymbol{\psi_{10}}, \boldsymbol{\psi_5}, \boldsymbol{\psi_5}, \boldsymbol{\psi_8}$

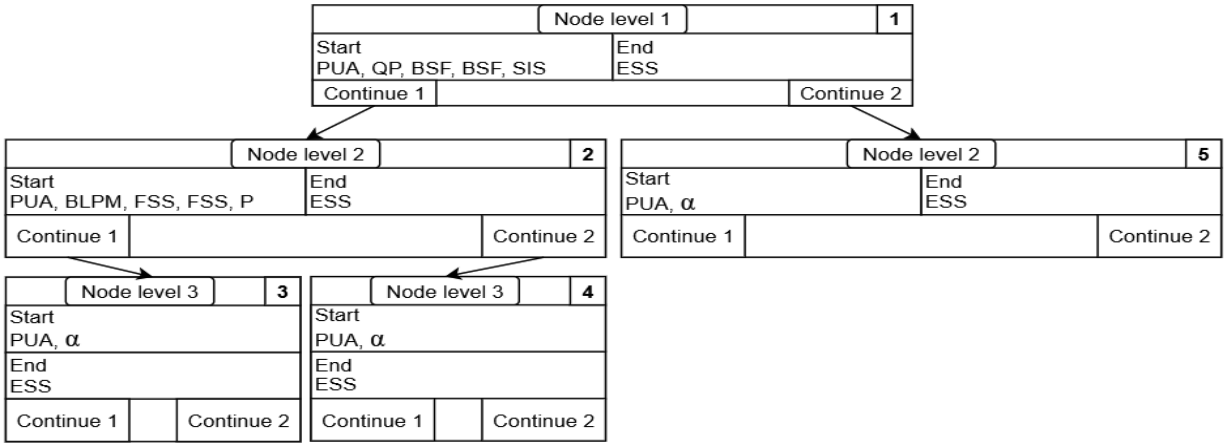


Рис. 3. Змінена форма після застосування правил $\boldsymbol{\psi_{14}}$, $\boldsymbol{\psi_1}$, $\boldsymbol{\psi_1}$, $\boldsymbol{\psi_7}$ щодо форми на рис. 2

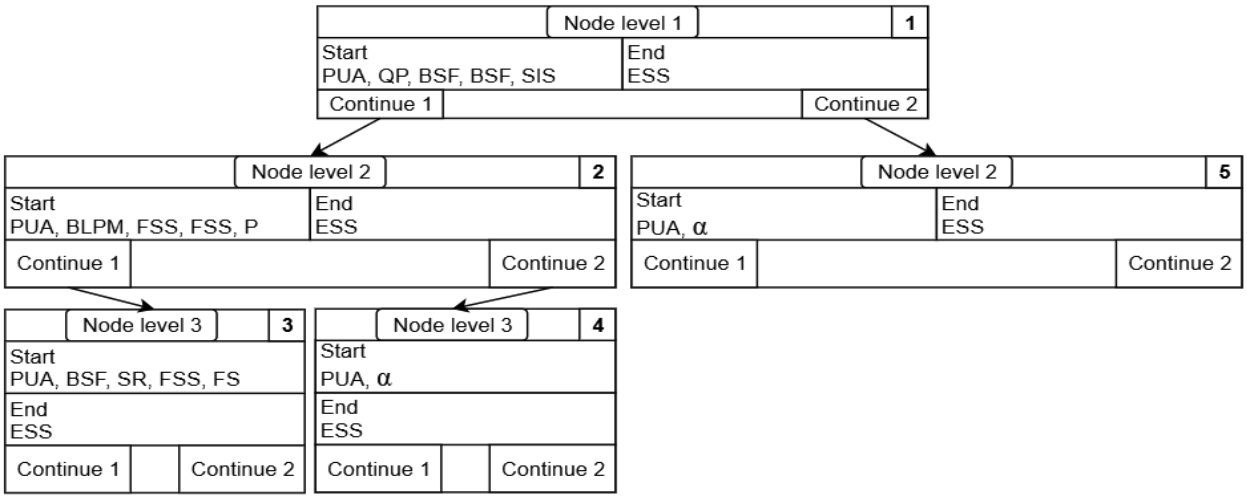


Рис. 4. Змінена форма після застосування правил $\boldsymbol{\psi_5}$, $\boldsymbol{\psi_{12}}$, $\boldsymbol{\psi_1}$, $\boldsymbol{\psi_9}$ щодо форми на рис 3

лідовності генів (рис. 2). Нерозкриті нетермінали лишились у блоках 2 та 5.

Наступними виконуються підстановки для вузла 2 (рис. 3). Обирається правило підстановки $\psi_{14}$. Додається ген BLPM та встановлюються значення атрибутів $nG = 1; pGU = true;$. Далі обираються два правила $\psi_1$. Додаються гени FSS і встановлюються значення атрибутів $nG = 3; pGU = false;$.

Обирається останнє правило $\psi_7$. Додається ген і встановлюються значення атрибутів cD=2; pGU=false; nG=0;. Додаються два вузли-нащадки під номерами 3 і 4 до поточного вузла, також гени і нетермінали.

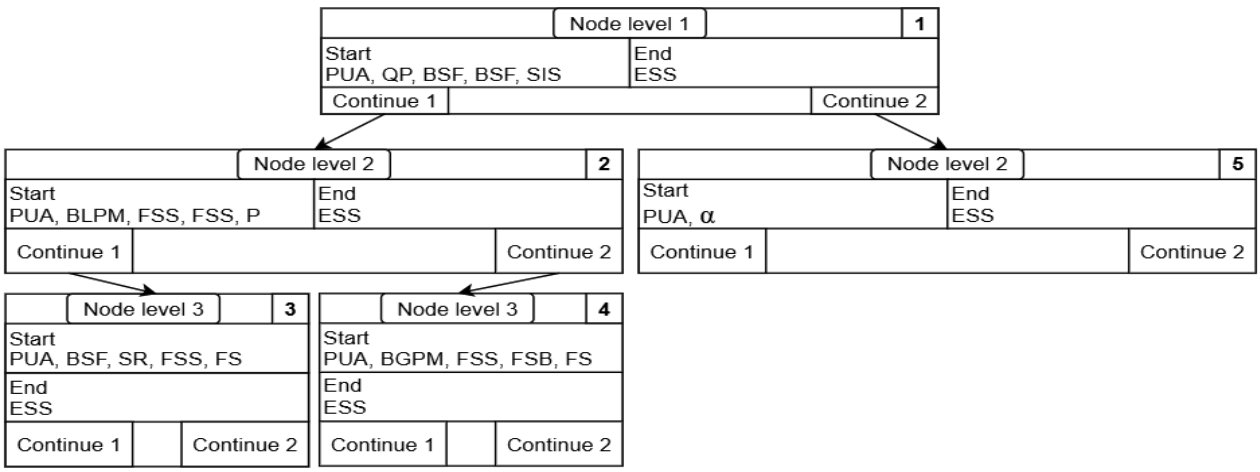


Рис. 5. Змінена форма після застосування правил $\boldsymbol{\psi_{14}}$, $\boldsymbol{\psi_1}$, $\boldsymbol{\psi_2}$, $\boldsymbol{\psi_9}$ щодо форми на рис. 4

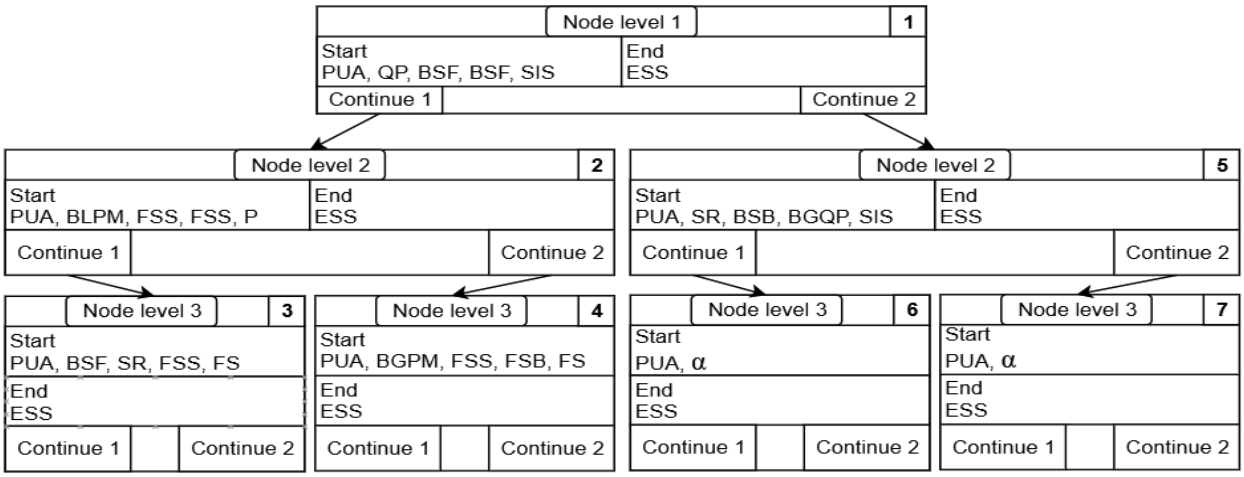


Рис. 6. Змінена форма після застосування правил $\boldsymbol{\psi_{12}}$, $\boldsymbol{\psi_6}$, $\boldsymbol{\psi_{15}}$, $\boldsymbol{\psi_8}$ щодо форми на рис. 5

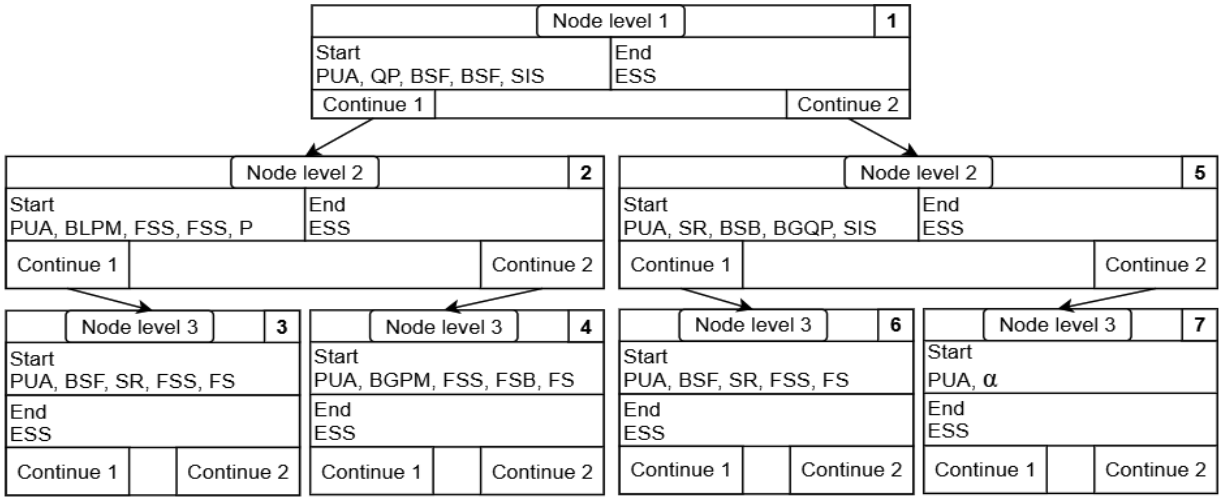


Рис. 7. Змінена форма після застосування правил $\boldsymbol{\psi_5}$, $\boldsymbol{\psi_{12}}$, $\boldsymbol{\psi_1}$, $\boldsymbol{\psi_5}$ щодо форми на рис. 6

Наступним виконуються підстановки для вузла 3 (рис. 4). Обирається правило підстановки $\psi_5$. Додається ген BSF, встановлюються атрибути $nG = 1$; $pGU = false$;. Під час виконання підстановки за правилом $\psi_{12}$, до послідоності генів додається ген SR, встановлюються значення атрибутів $nG = 2; pGU = true$;. Далі обирається правило $\psi_1$, додається ген FSS, встановлюються значення атрибутів $nG = 3; pGU = false$;. Оскільки досягнута максимальна глибина хромосоми-дерева, гени розбиття стають недоступними і обирається правило фінального сортування $\psi_9$. Нові вузли не створюються і формування поточного вузла завершується.

Наступними виконуються підстановки для вузла 4 (рис. 5). Обираються правила підстановки $\psi_{14}$, $\psi_1$, $\psi_2$. Оскільки досягнута максимальна глибина хромосоми-дерева, гени розбиття стають недоступні і обирається правило фінального сортування $\psi_9$. Нові вузли не створюються і формування поточного вузла завершується.

Виконуються підстановки для вузла 5 (рис. 6). Обираються правила підстановки $\psi_{12}$, $\psi_6$, $\psi_{15}$, $\psi_8$. Додаються два вузли-нащадки під номерами 6 і 7 до поточного вузла, додаються гени і нетермінали.

Виконуються підстановки для вузла 6 (рис. 7). Обираються правила підстановки $\psi_5$, $\psi_{12}$, $\psi_1$. Оскільки досягнута максимальна глибина хромосоми-дерева, гени розбиття стають недоступними і обирається правило фінального сортування $\psi_9$. Нові вузли не створюються і формування поточного вузла завершується.

Виконуються підстановки для вузла 7 (рис. 8). Обираються правила підстановки $\psi_{15}$, $\psi_2$, $\psi_6$. Оскільки досягнута максимальна глибина хромосоми-дерева, гени розбиття стають недоступні і обирається правило фінального сортування $\psi_9$. Нові

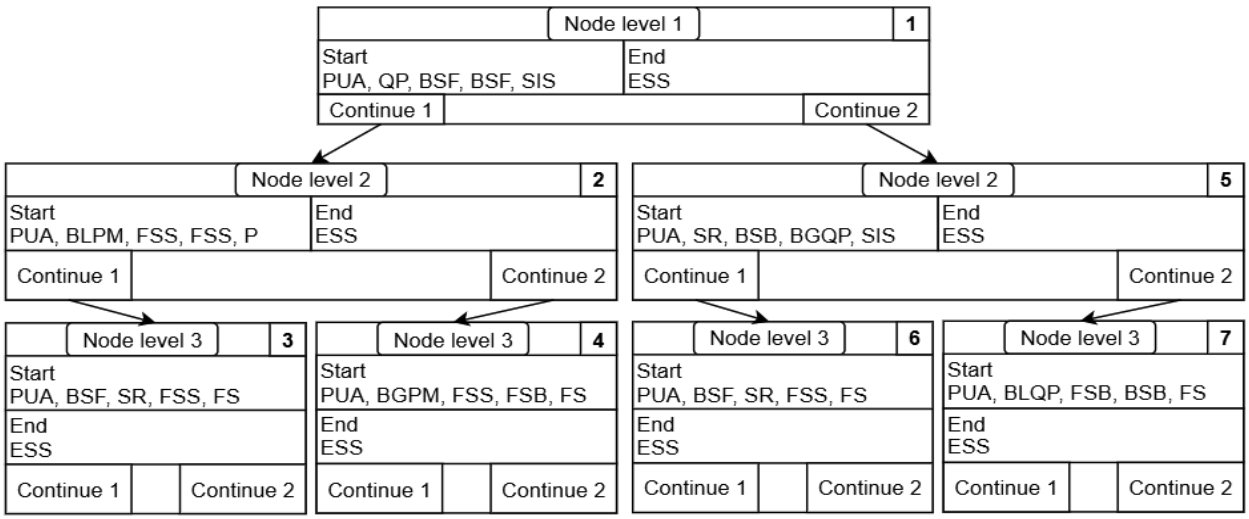


Рис. 8. Змінена форма після застосування правил $\boldsymbol{\psi_{15}}, \boldsymbol{\psi_{2}}, \boldsymbol{\psi_{6}}, \boldsymbol{\psi_{15}}$ щодо форми на рис. 7

вузли не створюються і формування поточного вузла завершується.

## Використання хромосом генетичним алгоритмом

Для застосування генетичного алгоритму необхідно сформувати початкову популяцію хромосом. Потрібне перетворення хромосоми у текст програми і виконання сортування експериментальних даних кожним алгоритмом-індивідом популяції. Здійснюється пошук певної кількості кращих алгоритмів. Кращим вважається алгоритм, що виконує сортування за менший проміжок часу. Потрібен відбір певної кількості кращих хромосом і додавання їх до наступної популяції. Далі - формування нових хромосом із застосуванням механізмів схрещення і мутацій, а також додавання нових.

Далі представлений скелет програми.

*[Крок 1. Налаштування параметрів експерименту]*
*[Крок 2. Генерація популяції]*
*[Крок 3. Генерація вхідних даних]*
*[Крок 4. Оцінка хромосоми]*
*[Крок 4.1. Декодування хромосоми в код алгоритму сортування]*
*[Крок 4.1.1. Для кожного гена]*
*[Крок 4.1.1.1. Отримати код, що відповідає переданому гену]*
*[Крок 4.1.1.2. Додати код гена до коду алгоритму сортування]*
*[Крок 4.2. Побудова програми для запуску згенерованого алгоритму сортування]*
*[Крок 4.3. Запуск програми, яка сортує вхідні дані та вимірює час виконання]*
*[Крок 5. Отримання результатів оцінки]*
*[Крок 6. Генерація нової популяції або завершення експерименту]*

Приклад коду формування тексту програми за геном QP у кроці 4.1.1.1.

```
std::tie(min, max) = Utilities::Find-
MinMax(arr + s, e - s + 1);
if (min == max)
{
 isSorted = true;
}
if (!isSorted)
{
 auto range = max - min;
 auto maxIndex = e - s;
 auto maxStepsQP = std::min(std::max((e -
s + 1) / 1000, 10), 100);
 for (int i = s; i <= e; ++i)
 {
  int j = 0;
  while (j++ < maxStepsQP)
  {
   int predictedPlace = double(arr[i] - min)
/ range * maxIndex + s;
   if (i == predictedPlace || arr[i] ==
arr[predictedPlace])
   {
    break;
   }
  std::swap(arr[i], arr[predictedPlace]);
  }
}
```

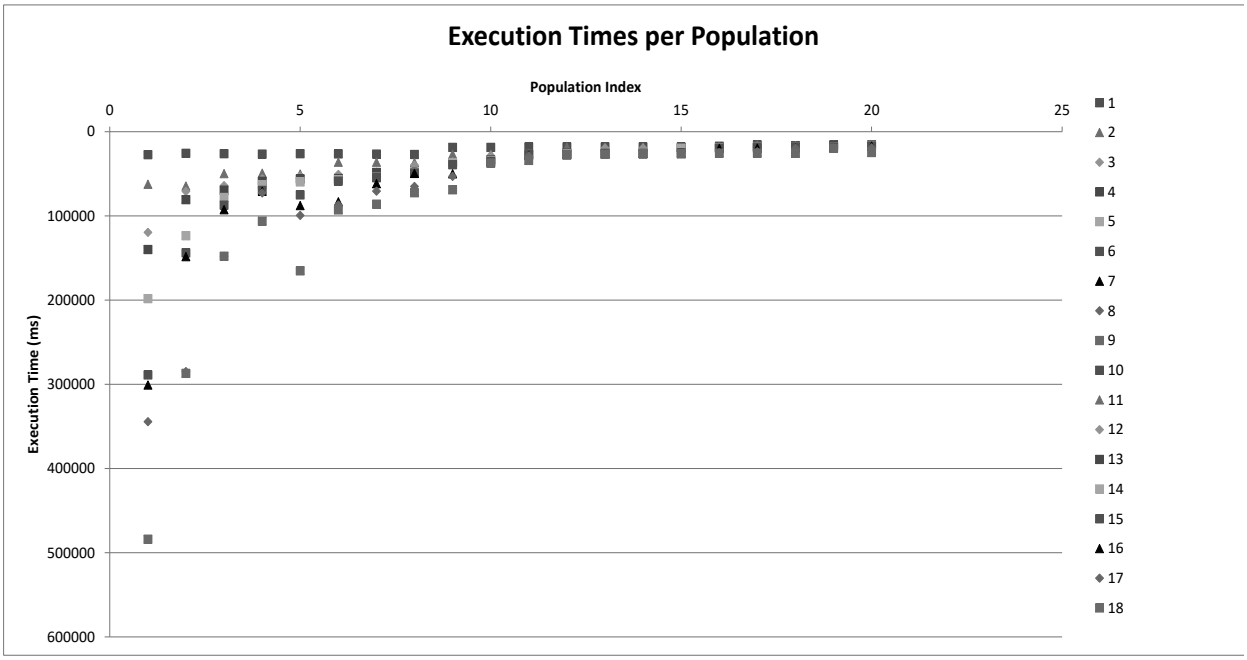


Рис. 9. Зміна часу сортування у різних поколіннях хромосом

У тексті програми: *s* – індекс першого елемента сортованого підмасива; *e* – індекс останнього елемента сортованого підмасива; *maxStepsQP* – максимальна кількість передбачень для кожного елемента; *min*, *max* – значення найменшого і найбільшого елементів сортованого підмасиву; *arr* – сортований масив.

## Експериментальні дослідження

У ході роботи було проведено багато експериментів із різними типами і обсягами даних за різних початкових умов і параметрів завершення. Далі представлено один із експериментів еволюційного вибору кращого алгоритму сортування спецефічних даних.

Специфікація полягає у наступному. Експеримент проводився на даних обсягом один мільйон елементів. Масив заповнювався випадковими числами у діапазоні від нуля до одного мільйона, потім виконувалось повне сортування даних. Далі у повністю відсортованому масиві вибирались N ділянок з сумарною довжиною L. У межах кожної ділянки випадковим чином вибирались 2 індекси, і елементи за обраними індексами мінялись місцями, процес повторювався кількість разів відповідну довжині ділянки. Наведені експерименти виконувались на ноутбуці із технічними характеристиками, представленими у табл.1.

Таблиця 1.

Технічні засоби експерименту

| Характеристики центрального процесору | |
|---|---|
| Поле | Значення |
| Тип | Intel Core i7-12650H (12th Gen) |
| Кількість ядер | 10 cores (6 Performance + 4 Efficiency) |
| Кеш L1 | 864 KB |
| Кеш L2 | 9.5 MB |
| Кеш L3 | 24 MB |
| Характеристики оперативної пам’яті | |
| Поле | Значення |
| Тип | DDR5-4800 |
| Розмір модулю | 8 ГБ |
| Кількість модулів | 2 x 8 ГБ |

На рис. 9 представлені результати експерименту. До останньої популяції додано 4 загальновідомі алгоритми для порівняння: швидке, злиттям, вставками і сортування вибором. Кращий алгоритм знаходиться праворуч вгорі, нижче йдуть усі інші алгоритми у порядку погіршення часової ефективності.

Далі представлені п’ять кращих хромосом і усі загальновідомі алгоритми, позначені на графіку числами відповідно:

1. SR_SIS_PUA_P_PUA_BSB_BSF_FS_ESS_PUA_FSS_BSB_BSB_FS_ESS_ESS_PUA_P_PUA_FSB_FS_ESS_PUA_FS_ESS_ESS_ESS
2. SR_SIS_PUA_P_PUA_BSB_FSS_FS_ESS_PUA_FSB_BSB_BSB_FS_ESS_ESS_PUA_P_PUA_FSB_FS_ESS_PUA_FS_ESS_ESS_ESS
3. SR_SIS_PUA_P_PUA_BSB_FSS_FS_ESS_PUA_FSS_BSB_BSB_FS_ESS_ESS_PUA_P_PUA_FSB_FS_ESS_PUA_FS_ESS_ESS_ESS
4. SR_SIS_PUA_P_PUA_BSB_FSS_FS_ESS_PUA_FSS_BSB_BSB_FS_ESS_ESS_PUA_P_PUA_FSB_FS_ESS_PUA_FS_ESS_ESS_ESS
5. QuickSort

   …
14. HeapSort

   …
18. MergeSort

   …
20. InsertionSort

На графіку надано час сортування різних алгоритмів. Перші чотири алгоритми (позиції 1–4) сформовані генетичним алгоритмом на основі підходу, представленого в даній роботі. Час сортування порівнюється з класичними алгоритмами сортування, які розмістились на позиціях 5, 14, 18 та 20.

Результати демонструють, що конструйовані алгоритми, представлені хромосомами, здатні перевершити традиційні за швидкістю виконання на тестових наборах даних. Підтверджують доцільність використання нових алгоритмів у задачах, де критичним є час обробки великих масивів інформації.

## Висновки

У разі структурної адаптації алгоритмів досягається можливість гнучко перебудовувати їхню внутрішню структуру залежно від властивостей вхідних даних, цільових вимог чи обмежень обчислювального середовища. Такий підхід дозволяє поєднувати фундаментальні алгоритмічні фрагменти у нові структури, що зберігають коректність і водночас покращують часові та ресурсні характеристики.

Розроблено підхід до формування хромосом, які відповідають алгоритмам сортування, сформованим із частин класичних алгоритмів сортування (таких як Bubble Sort, Quick Sort, Merge Sort) і деяких передобробок у вигляді структур, придатних для генетичних та еволюційних обчислень.

Конструктивно-продукційне моделювання продемонструвало ефективність у формуванні хромосом, що забезпечує можливість структурної адаптації алгоритмів для специфічних вхідних даних.

Запропонована модель забезпечує гнучкість у генерації хромосом з урахуванням різних критеріїв оптимізації, таких як час виконання. Структура кодованого дерева забезпечує уніфікований підхід до процесу генерації послідовності генів для кожного вузла за допомогою рекурсії. Таким чином уможливлюється генерування хромосом будь-якої довжини.

Проведені експерименти підтвердили, що структуровані хромосоми, сформовані за допомогою продукційних правил, демонструють вищу ефективність у задачах еволюційного пошуку порівняно з випадковою ініціалізацією.

Отримані результати можуть бути використані для автоматизованого синтезу алгоритмів, оптимізації програмного коду та побудови інтелектуальних систем, що навчаються.

## Література

1. C. A. R. Hoare. Quicksort. The Computer Journal. – Vol. 5, No. 1. – pp. 10–16. – 1962.
2. D. E. Knuth. Sorting by Exchanging. The Art of Computer Programming. – Vol. 3: Sorting and Searching. – 1st ed. – Addison-Wesley. – pp. 110–111. – 1973.
3. N. Auger, V. Jugé, C. Nicaud, C. Pivoteau. On the worst-case complexity of Timsort. 26th Annual European Symposium on Algorithms (ESA 2018). – LIPIcs, Vol. 112. – pp. 4:1–13. – 2018. – Available: https://arxiv.org/abs/1805.08612
4. D. R. Musser. Introspective Sorting and Selection Algorithms. Software: Practice and Experience. – Vol. 27, No. 8. – pp. 983–993. – 1997.
5. T. B. Martyniuk, B. I. Krukivskyi. Peculiarities of the parallel sorting algorithm with rank

formation. Cybernetics and Systems Analysis. – Vol. 58, No. 1. – pp. 24–28. – 2022. – DOI: https://doi.org/10.1007/s10559-022-00431-8.

6. M. Dietzfelbinger. How Good Is Multi-Pivot Quicksort?. ArXiv, Cornell University. – 2015.
7. A. S. Mohammed, Ş. E. Amrahov, F. V. Çelebi. Bidirectional Conditional Insertion Sort algorithm: An efficient progress on the classical insertion sort. Future Generation Computer Systems. – Vol. 71. – pp. 102–112. – June 2017.
8. V. Jugé. Adaptive shivers sort: an alternative sorting algorithm. ACM Transactions on Algorithms. – Vol. 20, No. 4. – pp. 1–55. – August 2024.
9. O. Appiah, E. M. Martey. Magnetic Bubble Sort Algorithm. International Journal of Computer Applications. – Vol. 122, No. 21. – pp. 24–28. – 2015. – DOI: 10.5120/21850-5168
10. A. Alotaibi, A. Almutairi, H. Kurdi. Onebyone (obo): A fast sorting algorithm. Procedia Computer Science. – Vol. 175. – pp. 270–277. – 2020.
11. F. M. Isa, W. N. M. Ariffin, M. S. Jusoh, E. P. Putri. A Review of Genetic Algorithm: Operations and Applications. Journal of Advanced Research in Applied Sciences and Engineering Technology. – 2020. – DOI: 10.37934/araset.40.1.134
12. Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. – 3rd ed. – Springer. – 1996.
13. В.І. Шинкаренко, В.М. Ільман. Конструктивно-продукційні структури та їх граматичні інтерпретації. І. Узагальнена формальна конструктивно-продукційна структура . Кібернетика і системний аналіз. – 2014. – Т. 50, № 5. – С. 8–16.
14. В.І. Шинкаренко, А.Ю. Дорошенко, О.А. Яценко, В.В. Разносілін, К.К. Галанін, Двокомпонентні алгоритми сортування, Проблеми програмування, 2022, № 3-4. С. 32-41. doi: 10.15407/pp2022.03-04.032.
15. В.І. Шинкаренко, О.В. Макаров, Дослідження ефективності деяких детермінованих методів передобробки сортування даних, Проблеми програмування, 2023, № 4, С. 3-14. doi: 10.15407/pp2023.04.003.
16. V. I. Shynkarenko, O. V. Makarov. Structural Adaptation of Sorting Algorithms Based on Constructive Fragments. CEUR Workshop Proceedings. – Vol. 3806. – pp. 16–29. – 2024.

## References

1. C. A. R. Hoare. Quicksort. The Computer Journal. – Vol. 5, No. 1. – pp. 10–16. – 1962.
2. D. E. Knuth. Sorting by Exchanging. The Art of Computer Programming. – Vol. 3: Sorting and Searching. – 1st ed. – Addison-Wesley. – pp. 110–111. – 1973.
3. N. Auger, V. Jugé, C. Nicaud, C. Pivoteau. On the worst-case complexity of Timsort. 26th Annual European Symposium on Algorithms (ESA 2018). – LIPIcs, Vol. 112. – pp. 4:1–13. – 2018. – Available: https://arxiv.org/abs/1805.08612
4. D. R. Musser. Introspective Sorting and Selection Algorithms. Software: Practice and Experience. – Vol. 27, No. 8. – pp. 983–993. – 1997.
5. T. B. Martyniuk, B. I. Krukivskyi. Peculiarities of the parallel sorting algorithm with rank formation. Cybernetics and Systems Analysis. – Vol. 58, No. 1. – pp. 24–28. – 2022. – DOI: https://doi.org/10.1007/s10559-022-00431-8
6. M. Dietzfelbinger. How Good Is Multi-Pivot Quicksort?. ArXiv, Cornell University. – 2015.
7. A. S. Mohammed, Ş. E. Amrahov, F. V. Çelebi. Bidirectional Conditional Insertion Sort algorithm: An efficient progress on the classical insertion sort. Future Generation Computer Systems. – Vol. 71. – pp. 102–112. – June 2017.
8. V. Jugé. Adaptive shivers sort: an alternative sorting algorithm. ACM Transactions on Algorithms. – Vol. 20, No. 4. – pp. 1–55. – August 2024.
9. O. Appiah, E. M. Martey. Magnetic Bubble Sort Algorithm. International Journal of Computer Applications. – Vol. 122, No. 21. – pp. 24–28. – 2015. – DOI: 10.5120/21850-5168
10. A. Alotaibi, A. Almutairi, H. Kurdi. Onebyone (obo): A fast sorting algorithm. Procedia Computer Science. – Vol. 175. – pp. 270–277. – 2020.
11. F. M. Isa, W. N. M. Ariffin, M. S. Jusoh, E. P. Putri. A Review of Genetic Algorithm: Operations and Applications. Journal of Advanced Research in Applied Sciences and Engineering Technology. – 2020. – DOI: 10.37934/araset.40.1.134
12. Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. – 3rd ed. – Springer. – 1996.
13. V. I. Shynkarenko, V. M. Ilman. Constructive-Synthesizing Structures and Their Grammatical Interpretations. I. Generalized Formal Constructive-Synthesizing Structure.

Cybernetics and Systems Analysis. – Vol. 50, No. 5. – pp. 8–16.
14. V. I. Shinkarenko, A. Yu. Doroshenko, O. A. Yatsenko, V. V. Raznosilin, K. K. Galanin. Bicomponent sorting algorithms. Problems of Programming. – 2022. – No. 3–4. – pp. 32–41. – DOI: 10.15407/pp2022.03-04.032
15. V. I. Shinkarenko, O. V. Makarov. Study of the effectiveness of some deterministic preprocessing methods of data sorting. Problems of Programming. – 2023. – No. 4. – pp. 3–14. – DOI: 10.15407/pp2023.04.003
16. V. I. Shynkarenko, O. V. Makarov. Structural Adaptation of Sorting Algorithms Based on Constructive Fragments. CEUR Workshop Proceedings. – Vol. 3806. – pp. 16–29. – 2024.

Одержано: 12.10.2025
Внутрішня рецензія отримана: 18.10.2025
Зовнішня рецензія отримана: 19.10.2025

***Про авторів:***

*Шинкаренко Віктор Іванович,*
доктор технічних наук, професор,
https://orcid.org/0000-0001-8738-7225
E-mail: shinkarenko_vi@ua.fm

*Макаров Олексій Вікторович,*
аспірант,
https://orcid.org/0009-0003-0921-155X
E-mail: makarovov@hotmail.com

***Місце роботи авторів:***

Український державний університет
науки і технологій,
49010, Україна, Дніпро,
вул. Академіка Лазаряна, 2.
E-mail: office@ust.edu.ua