УДК 004.4 https://doi.org/10.15407/pp2026.01.040

*В.І. Шинкаренко,* О.*В. Макаров*

# КОНСТРУКТИВНО-ПРОДУКЦІЙНЕ ФОРМУВАННЯ ПРОГРАМ СОРТУВАННЯ, АДАПТОВАНИХ ГЕНЕТИЧНИМ АЛГОРИТМОМ

У попередніх роботах представлені механізми конструктивно-продукційного моделювання для адаптації алгоритмів сортування. У зв'язку з цим виникли задачі перетворення хромосом генетичного алгоритму на текст програм сортування для подальшого застосування, оцінки можливостей еволюційного розвитку. Розглядається підхід до перетворення хромосом, що кодують алгоритми сортування на тексти програм готових до застосування у реальних умовах. Розроблено конструктор-трансформер, який реалізує перетворення хромосоми-дерева на лінійну послідовність генів. Інший конструктор-трансформер призначений для перетворення послідовності генів на код програми сортування. Наведено приклади послідовності обходу дерева-хромосоми, додавання генів до лінійної послідовності і формування тексту програми. Проведено експерименти із вхідними даними різної структури і обсягів. Результати експериментів підтвердили, що запропонована методика може бути використана для автоматичної генерації ефективних алгоритмів сортування. А застосування конструктивно-продукційного моделювання у сукупності із генетичним алгоритмом дозволяє ефективно виконувати структурну адаптацію алгоритмів.
Ключові слова: конструктивно-продукційне моделювання, алгоритми сортування, часова ефективність, генетичний алгоритм, програмне забезпечення, інформаційні технології.

*V.I. Shynkarenko, O.V. Makarov*

# CONSTRUCTIVE-SYNTHESIZING PRODUCTION OF SORTING PROGRAMS ADAPTED BY GENETIC ALGORITHM

In previous works, the mechanisms of constructive-synthesizing modeling for the adaptation of sorting algorithms were presented. In this regard, the task of transforming the chromosomes of a genetic algorithm into the text of sorting programs for further application, evaluation and the possibility of evolutionary development arose. An approach to the transformation of the chromosome encoding a sorting algorithm into the text of a sorting program ready for use in real conditions is considered. A transformer constructor has been developed that implements the transformation of a chromosome tree into a linear sequence of genes. Another transformer constructor is designed to transform a sequence of genes into the code of a sorting program. Examples of the sequence of traversing a chromosome tree, adding genes to a linear sequence and forming the text of the program are given. Experiments were conducted with input data of different structures and volumes. The results of the experiments confirmed that the proposed method can be used for the automatic generation of effective sorting algorithms. And the use of constructive-synthesizing modeling in conjunction with a genetic algorithm allows for the effective structural adaptation of algorithms.
Key words: constructive-synthesizing modeling, sorting algorithm, time efficiency, genetic algorithm, software, information technology.

## Вступ

Сортування – один із фундаментальних будівельних блоків алгоритмічної інженерії. Класичні порівняльні методи, такі як quicksort [1], mergesort [2] і heapsort [3] формують «базову лінійку» загального призначення, демонструючи очікувану обчислювальну складність $O(n \log n)$ у середньому чи в гіршому випадку. Непорівняльні підходи – counting sort [4], radix sort [5], bucket sort [6] – досягають лінійних меж $O(n)$ за умови обмеженого діапазону або специфічного розподілу ключів у вхідних даних. Не дивлячись на класичні алгоритми, які розробляються давно, зараз продовжується різностороння діяльність з їхнього вдосконалення.

Одним із яскравих прикладів сучасного прогресу є робота "Fast and Simple Sorting Using Partial Information" [7], де розглядається проблема сортування з частко-



 

вою інформацією, коли деякі попарні порівняння вже відомі. Деяка кількість попередніх порівнянь доступні, і завдання полягає в тому, щоб використати мінімальну кількість нових порівнянь для повного відсортування даних.

Паралельно тривають дослідження алгоритмів, які використовують локальність даних та їхню кластерну структуру для прискорення сортування. Новітній представник цього напряму – Cluster Sort [8], запропонований як гібридний in-place підхід: елементи групуються (кластеризуються) з урахуванням просторової локальності, після чого в межах кластерів застосовуються ефективні процедури впорядкування.

Наразі запропоновано декілька видів детермінованих передобробок даних, призначених для сортування [9, 10]. Проведені експерименти, які показали, що використання передобробок може підвищити часову ефективність алгоритмів сортування. Тому доцільно використати алгоритми передобробки у процесі формування алгоритмів сортування. Це розширює можливості для конструювання, значно збільшує варіативність сформованих алгоритмів, однак привносить додаткові правила та обмеження. Через те, що алгоритми передобробки мають схожий принцип дії, недоцільне послідовне використання кількох алгоритмів передобробки, що унеможливлює використання двох алгоритмів передобробки одночасно.

У статті [11] була запропонована конструктивна модель хромосоми, в якій алгоритм сортування кодується як набір елементарних складових (production-based fragments), що можуть комбінуватися, розширюватися та еволюціонувати. Подання алгоритму у вигляді конструктора, де частини алгоритмів сортування виступають як будівельні блоки, дозволяє:

- поєднувати фрагменти різних класичних алгоритмів сортування;
- автоматично створювати гібридні алгоритми;
- адаптувати структуру алгоритму під конкретні властивості даних і умови застосування.

У продовження попередньої статті [12], де було сформовано підхід до конструктивно-продукційного моделювання хромосом, застосовано розроблену методику для дослідження та структурної адаптації алгоритмів сортування. За допомогою конструктора-трансформера, виконується декодування хромосоми у послідовність генів. Конструктор формування тексту програм перетворює отриману послідовність на код програми обраною мовою програмування.

Виконані експерименти у яких конструйовані алгоритми сортування застосовуються у різних обсягах і даних. За допомогою генетичного алгоритму [13] виконується адаптація структури сформованих алгоритмів до специфічних характеристик вхідних даних, що дозволяє отримувати алгоритми з кращими часовими показниками у заданих умовах використання.

## Конструктор-трансформер

Конструктор-трансформер реалізує перетворення дерева-хромосоми із закодованими складовими алгоритму сортування на масив (послідовність) генів.

Визначимо спеціалізацію конструктора $C_{CD}$ на основі узагальненого коструктора C [14]:

$$C = \langle \mathrm{M}, \Sigma, \Lambda \rangle \ {}_{s}\!\longmapsto\ C_{CD} = \langle \mathrm{M}_{CD}, \Sigma_{CD}, \Lambda_{CD} \rangle, \quad (1)$$

де M – неоднорідний розширюваний носій структури, Σ – сигнатура, що складається з множин операцій зв'язування, підстановки і виводу, операцій над атрибутами і відношення підстановки, Λ – інформаційне забезпечення конструювання, яке включає онтологію, цілі, правила і обмеження конструювання; $\mathrm{M}_{CD}$ – носій, що включає термінальний ($T_{CD}$) і нетермінальний ($N_{CD}$) алфавіт, а також множину правил продукції Ψ з правилами $\psi_i = \langle s_{\mathrm{i}}, g_{\mathrm{i}} \rangle \in \Psi$, де i – номер правила, $s_{\mathrm{i}}$ – послідовність відношень підстановки та $g_{\mathrm{i}}$ – операцій над атрибутами. $\Sigma_{CD}$ – операції та відносини на елементах $\mathrm{M}_{CD}$. Інформаційне забезпечення конструювання (ІЗК) $\Lambda_{CD} \supset \Lambda$.

Конструктор $\mathrm{C}_{CD}$ містить правила підстановки $\psi_i = \langle \langle s_{i,1}, s_{i,2} \rangle, \langle g_{\mathrm{i},1}, g_{\mathrm{i},2} \rangle \rangle$, де

відношення $s_{i,1}$ реалізує розбір хромосоми з використанням стеку для збереження проміжних результатів. Відношення $s_{i,2}$ призначене для формування списку генів.

Операції виконуються у наступній послідовності: спочатку виконуються операції над атрибутами $g_{i,1}$, потім операції підстановки $s_{i,1}$ та $s_{i,2}$, і в кінці – $g_{i,2}$.

Нетермінальний алфавіт $N = \{\alpha_i\}$ складається з допоміжних символів.

Термінальний алфавіт – множина генів хромосоми, що трансформується. Наприклад, ген швидкої передобробки QP – передбачення позиції елемента у відсортованому масиві і перестановка, ген, який відповідає одному проходу алгоритму сортування бульбашкою у зворотному порядку – BSB. Повний перелік генів представлено у [12]. Термінали 'start' та 'end' відповідають початковій і кінцевій послідовностям генів вузла – 'Start' і 'End' на рис. 1.

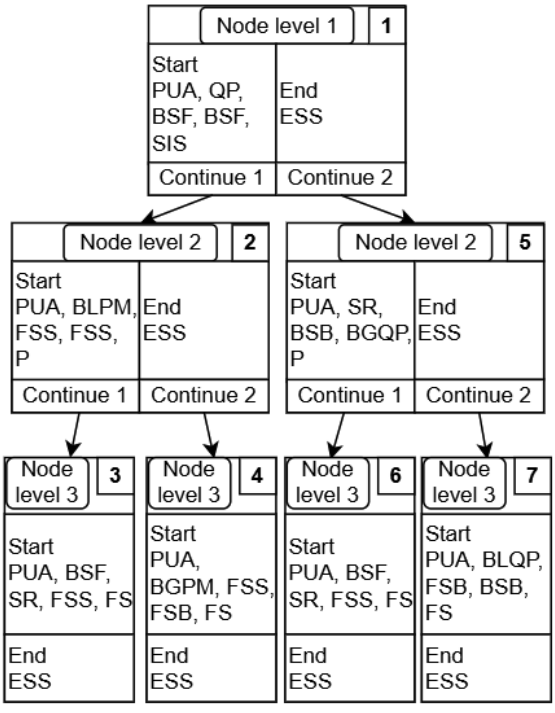


Рис. 1. Приклад хромосоми у вигляді дерева

Кожний вузол дерева на рис. 1 має початкову і кінцеву послідовності генів а також вказівники на вузли-нащадки. Листкові вузли мають порожні вказівники, тобто не мають нащадків.

У процесі трансформації дерева-хромосоми операції підстановки реалізують додавання терміналів (генів) до результуючої послідовності (масиву). Для обходу дерева вузли додаються до допоміжної структури – стеку, використання якого замінює рекурсію.

Атрибути поточної форми:

- current – вказівник на поточний вузол дерева-хромосоми;
- stack – стек вказівників на вузли дерева, які не були трансформовані;
- stackSize – розмір стеку вказівників на вузли дерева.

Розглянемо особливості відношень підстановки.

При використанні відношення $\alpha \to S \downarrow \beta \cdot \alpha$ у процесі конструювання $\beta$ може бути додано до стеку S. Подальші підстановки будуть продовжено для $\alpha$.

Використання відношення $\alpha \ {}_{\tau}\!\to \beta \uparrow S \cdot \alpha$ дає можливість дістати верхній елемент із стеку S і покласти його у $\beta$, якщо атрибут доступності $\tau$ відношення підстановки дорівнює true.

Виконаємо конкретизацію конструктора $\mathrm{C}_{CD}$.

$$C_{CD} = \langle \mathrm{M}_{CD}, \Sigma_{CD}, \Lambda_{CD} \rangle \ {}_{K}\!\longmapsto\ C_{CDK} = \langle \mathrm{M}_{CDK}, \Sigma_{CDK}, \Lambda_{CDK} \rangle, \quad (2)$$

де $\mathrm{M}_{CDK} = \mathrm{M}_{CD}, \Sigma_{CDK} = \Sigma_{CD}, \Lambda_{CDK} \supset \Lambda_{CD}$.

Інформаційне забезпечення конструювання $\Lambda_{CDK}$ розширене наступними правилами.

Перше правило $\psi_1 = \langle \langle s_{1,1}, s_{1,2} \rangle, \langle g_{1,1}, g_{1,2} \rangle \rangle$ реалізує додавання кореневого вузла дерева-хромосоми до стеку вузлів з яких ще не були скопійовані послідовності генів. Буде виконуватись тільки один раз на початку процесу трансформації.

$$\begin{gathered} s_{1,1} = \langle \alpha \to stack \downarrow root \downarrow chromosome \cdot \alpha \rangle; \\ s_{1,2} = \langle \varepsilon \rangle; \\ g_{1,1} = \langle \varepsilon \rangle; \\ g_{1,2} = \langle =: (stackSize, 1) \rangle \end{gathered} \quad (2)$$

Правило $\psi_2 = \langle \langle s_{2,1}, s_{2,2} \rangle, \langle g_{2,1}, g_{2,2} \rangle \rangle$ реалізує діставання вузла зі стеку вузлів і додавання початкової і кінцевої послідовностей генів до послідовності терміналів.

$$
\begin{aligned}
s_{2,1} &= \langle \alpha \xrightarrow{\tau_1} current \uparrow stack \cdot \alpha \rangle; \\
s_{2,2} &= \langle \rho \rightarrow start \downarrow current \cdot \rho \cdot \\
&\quad end \downarrow current \rangle; \\
&> (f_1, \text{stackSize}, 0), \\
g_{2,1} &= \langle == (f_2, current, null), \rangle; \\
&\wedge (\tau_1, f_1, f_2) \\
g_{2,2} &= \langle -(stackSize, stackSize, 1) \rangle
\end{aligned} \tag{3}
$$

Третє правило $\psi_3 = \langle \langle s_{3,1}, s_{3,2} \rangle, \langle g_{3,1}, g_{3,2} \rangle \rangle$ реалізує додавання вузлів нащадків поточного вузла до стеку. У подальшому вузли будуть діставатись зі стеку у зворотному порядку, тому спочатку додається правий, а потім лівий вузол.

$$
\begin{aligned}
s_{3,1} &= \\
&\left\langle \begin{matrix} \alpha \xrightarrow{\tau_1} stack \downarrow right \downarrow current \cdot \alpha \\ \alpha \xrightarrow{\tau_1} stack \downarrow left \downarrow current \cdot \alpha \end{matrix} \right\rangle; \\
s_{3,2} &= \langle \varepsilon \rangle; \\
g_{3,1} &= \langle ! = (\tau_1, left \downarrow \\
&\quad current, null) \rangle; \\
g_{3,2} &= \left\langle \begin{matrix} +(stackSize, stackSize, 2) \\ =: (current, null) \end{matrix} \right\rangle
\end{aligned} \tag{4}
$$

Початкові умови конструювання: хромосома деревоподібної структури, де у кожного вузла є початкова і кінцева послідовності генів, і всі вузли, окрім листових, мають два вузли-нащадки.

Таблиця 1.

Приклад використання стеку у процесі трансформації хромосоми

| Стек вузлів | Вузол, який дістається зі стеку | Вузол, з якого копіюються гени і нащадки додаються до стеку |
|---|---|---|
| [1] | | |
| [] | 1 | |
| [5,2] | | 1 |
| [5] | 2 | |
| [5,4,3] | | 2 |
| [5,4] | 3 | |
| [5,4] | | 3 |
| [5] | 4 | |
| [5] | | 4 |
| [] | 5 | |
| [7,6] | | 5 |
| [7] | 6 | |
| [7] | | 6 |
| [] | 7 | |
| [] | | 7 |

У табл. 1 представлений процес обходу дерева з використанням стеку. Числа у таблиці відповідають номерам вузлів на рис. 1. Вузли будуть опрацьовані у наступній послідовності: 1, 2, 3, 4, 5, 6, 7.

Результатом трансформації буде наступна послідовність генів: [PUA, QP, BSF, BSF, SIS, PUA, BLPM, FSS, FSS, P, PUA, BSF, SR, FSS, FS, ESS, BGPM, FSS, FSB, FS, ESS, ESS, PUA, SR, BSB, BGQP, P, PUA, BSF, SR, FSS, FS, ESS, PUA, BLQP, FSB, BSB, FS, ESS, ESS, ESS].

## Конструктор формування тексту програм

Конструктор формування тексту програм (теж конструктор-трансформер) перетворює послідовність генів на текст програми обраною мовою програмування (у даній роботі – C++).

Визначимо спеціалізацію конструктора $\mathrm{C}_{PF}$ – моделі декодування послідовності генів, що кодує алгоритм сортування, у текст програми:

$$
\begin{aligned}
C = \langle \mathrm{M}, \Sigma, \Lambda \rangle \ &\xrightarrow{s} \ C_{PF} \\
&= \langle \mathrm{M}_{PF}, \Sigma_{PF}, \Lambda_{PF} \rangle,
\end{aligned} \tag{5}
$$

де $\mathrm{M}_{PF}$ – носій, що включає термінальний ($T_{PF}$) і нетермінальний ($N_{PF}$) алфавіт, а також множину правил продукції $\Psi$ з правилами $\psi_i = \langle s_\mathrm{i}, g_\mathrm{i} \rangle \in \Psi$, де i – номер правила, $s_\mathrm{i}$ – послідовність відношень підстановки, $g_\mathrm{i}$ – послідовність операцій над атрибутами. $\Sigma_{PF}$ – операції та відносини на елементах $\mathrm{M}_{PF}$. ІЗК $\Lambda_{PF} \supset \Lambda$.

Нетермінали – гени хромосоми, яка кодує алгоритм сортування.

Термінали – текстові фрагменти програм обраною мовою програмування, які відповідають частинам відомих алгоритмів сортування, передобробок і допоміжних функцій, і кодуються генами. Відповідно до гена X будемо використовувати термінал $\mathrm{T_X}$. Наведемо список усіх терміналів: $T_\mathrm{S}$, $T_\mathrm{E}$, $T_{\mathrm{B}SF}$, $T_{\mathrm{B}SB}$, $T_{\mathrm{F}SB}$, $T_{\mathrm{F}SS}$, $T_{SEEI}$, $T_{\mathrm{SE}I}$, $T_{\mathrm{SI}S}$, $T_\mathrm{P}$, $T_\mathrm{FS}$, $T_{\mathrm{P}UA}$, $T_\mathrm{ESS}$, $T_{\mathrm{M}L}$, $T_{\mathrm{M}R}$, $T_{\mathrm{Q}P}$, $T_{\mathrm{P}M}$, $T_\mathrm{SR}$, $T_{\mathrm{BL}QP}$, $T_{\mathrm{BL}PM}$, $T_{\mathrm{BG}QP}$, $T_{\mathrm{BG}PM}$.

Усі термінали $T^* \in T_{PF}$ мають атрибут $text \downarrow T^*$ – текст програми відповідний алгоритму, який кодується геном.

Атрибути терміналів $text \downarrow T_{\mathrm{S}}$ і $text \downarrow T_{\mathrm{E}}$ є незмінними (для любого сформованого алгоритму) частинами тексту програми, які додаються відповідно на початку і в кінці файлу. На початку ($T_{\mathrm{S}}$) включаються заголовкові файли із допоміжними функціями і типами даних, оголошуються допоміжні структури, початок реалізації функції сортування. Кінцева частина ($T_{\mathrm{E}}$) включає код злиття усіх пар із стеку відсортованих масивів для злиття.

Наступні термінали включають перевірку флагу відсортованості, оновлення значень змінних, що входять до глобального контексту, і код програми відповідно до:

- $T_{BSF}$ – одного проходу сортуванням бульбашкою в прямому напрямку;
- $T_{BSB}$ – одного проходу сортуванням бульбашкою у зворотному напрямку;
- $T_{FSB}$ – пошуку найбільшого елемента у невідсортованій частині масиву і перестановка на поточне місце;
- $T_{FSS}$ – пошуку найменшого елемента у невідсортованій частині масиву і перестановка на поточне місце;
- $T_{SEEI}$ – вставки поточного елемента у відсортовану частину у кінці масиву;
- $T_{SEI}$ – вставки поточного елемента у відсортовану частину на початку масиву;
- $T_{QP}$ – швидкої передобробки;
- $T_{PM}$ – передобробки із пам'яттю;
- $T_{SR}$ – передобробки із розворотом;
- $T_{BLQP}$ – блочної локальної швидкої передобробки;
- $T_{BLPM}$ – блочної локальної передобробки з пам'яттю;
- $T_{BGQP}$ – блочної глобальної швидкої передобробки;
- $T_{BGPM}$ – блочної глобальної передобробки з пам'яттю.

Вказані передобробки викладені в [10].

Атрибут $text \downarrow T_{SIS}$ представляє текст програми, який розділяє масив на дві частини, зберігає вказівники і розміри у стеку невідсортованих підмасивів (для подальшого окремного сортування) і у стеку масивів, для яких виконуватиметься злиття у кінці конструйованої функції.

Атрибут $text \downarrow T_{\mathrm{P}}$ є текстом, який обирає елемент, переставляє менші і більші елементи відповідно зліва і справа, зберігає вказівники і розміри частин масиву для подальшого сортування.

$text \downarrow T_{\mathrm{FS}}$ – атрибут, який є текстом загальновідомого алгоритму сортування. У поточній роботі використовується текст швидкого сортування.

Атрибути $text \downarrow T_{PUA}$ і $text \downarrow T_{ESS}$ представляють код допоміжних функцій. Перший дістає невідсортований масив зі стеку невідсортованих масивів для подальшого сортування. Другий додає закриті фігурні дужки до тексту програми у кількості, потрібній для успішної компіляції програми.

Атрибути терміналів $text \downarrow T_{ML}$ і $text \downarrow T_{MR}$ містять код, який оновлює значення змінних із глобального контексту і додає до стеку масивів для подальшого злиття відсортований підмасив (відповідно зліва і справа) та решту масиву.

Результатом закінчення конструювання буде цілісний текст програми, який включає конструйовану функцію сортування, допоміжні структури і функції.

Виконаємо конкретизацію конструктора $C_{PF}$:

$$C_{PF} = \langle \mathrm{M}_{PF}, \Sigma_{PF}, \Lambda_{PF} \rangle \underset{K}{\longmapsto} C_{PFK} = \langle \mathrm{M}_{PFK}, \Sigma_{PFK}, \Lambda_{PFK} \rangle, \quad (6)$$

де $\mathrm{M}_{PFK} = \mathrm{M}_{PF}$, $\Sigma_{PFK} = \Sigma_{PF} \cup \{+, :=, \downarrow, \uparrow\}$, $\Lambda_{PFK} \supset \Lambda_{CD}$.

Визначені наступні операції над атрибутами: +(a, b, c) – додавання аргументів b і c, результат зберігається у a; :=(a, b) –

присвоєння значення b змінній a; ↓(a, b) – додає елемент b у стек a; ↑(a, b) – дістає верхній елемент із стеку a, зберігає його значення у b.

Інформаційне забезпечення $\Lambda_{PFK}$ збагачено наступними положеннями.

Атрибути терміналів і нетерміналів поточної форми: nBrackets – поточна кількість відкритих фігурних дужок, відповідна кількість закритих дужок буде додана атрибутом $text \downarrow ESS$ для успішної компіляції коду програми; closingBrackets – стек лічильників відкритих фігурних дужок, верхній елемент відповідає поточній області видимості.

Початкові умови конструювання: послідовність (масив) генів, які кодують частини алгоритмів сортування і допоміжні операції.

Умови завершення конструювання: сформована програма сортування (поточна форма не містить нетерміналів).

Перше правило $\psi_1 = \langle \langle s_{1,1}, s_{1,2} \rangle, \langle g_{1,1}, g_{1,2} \rangle \rangle$ виконує підстановку постійної частини коду, однакової для усіх програм. Операції над атрибутами не виконуються.

$$\begin{gathered} s_{1,1} = \langle \sigma \to \gamma \rangle; \\ s_{1,2} = \langle \alpha \to text \downarrow T_s \cdot \alpha \cdot text \downarrow T_e \rangle; \\ g_{1,1} = \langle \varepsilon \rangle;\ g_{1,2} = \langle \varepsilon \rangle \end{gathered} \quad (7)$$

Для $\alpha$ будуть продовжені підстановки.

Наступне правило $\psi_2$ містить відношення підстановки програмного коду для гена PUA.

$$\begin{gathered} s_{2,1} = \langle \sigma \to \alpha \rangle; \\ s_{2,2} = \langle \alpha \to text \downarrow T_{PUA} \cdot \alpha \rangle; \\ g_{2,1} = \langle \varepsilon \rangle; \\ g_{2,2} = \langle \downarrow (\text{closingBrackets}, 0) \rangle \end{gathered} \quad (8)$$

Операції над атрибутами додають новий елемент до стеку лічильників фігурних дужок. Наступні операції над атрибутами, які збільшують лічильник фігурних дужок, будуть збільшувати останню додану змінну.

Правило $\psi_3$ містить відношення підстановки програмного коду для гена ESS.

$$\begin{gathered} s_{3,1} = \langle \sigma \to \alpha \rangle; \\ s_{3,2} = \langle \gamma \to text \downarrow T_{ESS} \cdot \gamma \rangle; \\ g_{3,1} = \langle \varepsilon \rangle; \\ g_{3,2} = \langle \uparrow (closingBranckets, nBrackets) \rangle \end{gathered} \quad (9)$$

Програмний код, що кодується геном ESS, виконує додавання фігурних дужок, що закриваються у кількості, рівній значенню атрибута nBrackets. Це потрібно для закриття областей видимості, відкритих додаванням попередніх фрагментів тексту програм.

Правило $\psi_4$ містить відношення підстановки для додавання програмного коду, що відповідає генам, які відкривають нову область видимості і збільшують лічильник дужок, які необхідно закрити. Для мінімізації кількості правил позначимо символом $Ter^*$ будь-який терміналген із множини {FSB, FSS, SEEI, SEI, BSB, BSF, P, QP, PM, SR, BLQP, BLPM, BGQP, BGPM}.

$$\begin{gathered} s_{4,1} = \langle \gamma \to Ter^* \cdot \gamma \rangle; \\ s_{4,2} = \langle \alpha \to text \downarrow Ter^* \cdot \alpha \rangle; \\ g_{4,1} = \langle \varepsilon \rangle; \\ g_{4,2} = \langle +(top \downarrow \text{closingBrackets}, top \downarrow \text{closingBrackets}, 1) \rangle \end{gathered} \quad (10)$$

Правило $\psi_5$ містить відношення підстановки для додавання програмного коду, що відповідає генам, які не відкривають нову область видимості. Позначимо символом $Ter^{**}$ будь-який терміналген із множини {SIS, ML, MR, FS}.

$$\begin{gathered} s_{5,1} = \langle \gamma \to Ter^{**} \cdot \gamma \rangle; \\ s_{5,2} = \langle \alpha \to text \downarrow Ter^{**} \cdot \alpha \rangle; \\ g_{5,1} = \langle \varepsilon \rangle; \\ g_{5,2} = \langle \varepsilon \rangle \end{gathered} \quad (11)$$

Результатом конструювання функції сортування буде не тільки текст безпосередньо функції сортування, а й текстове представлення коду всього файлу реалізації (.cpp). Відповідний заголовковий файл (.h) не змінюється залежно від хромосоми і створюється заздалегідь. Він включає оголошення функції сортування, що конструюється і має наступний вигляд:

```
#pragma once
void ConstructedSort(int* arr, int n);
```

Відповідний файл реалізації складається із двох частин: незмінної частини і реалізації функції сортування, що конструюється із хромосоми. У незмінній частині включається вищезгаданий заголовковий файл та заголовкові файли із допоміжними утилітами і типами даних. Оголошуються допоміжні структури і функції. І початок саме реалізації функції сортування, що конструюється, де оголошуються усі змінні-атрибути із початковими значеннями. Початкова незмінна частина файлу реалізації, яка відповідає терміналу $T_S$, має наступний вигляд (код у статті відрізняється форматуванням від коду програми, для економії місця):

```
// text ↲ T_S:
#include <algorithm>
#include <iostream>
#include <stack>
#include <vector>
#include "Sort/Sort.h"
#include "Utilities/Utilities.h"
void Merge(int* arr, int size1, int size2) {
std::vector<int> tempArray1;
tempArray1.assign(arr, arr + size1);
int indexOfMergedArray = 0; // Index of the first element in the merged array
int indexOfSubArrayOne = 0; // Index of the first element in the first sub-array
int indexOfSubArrayTwo = 0; // Index of the first element in the second sub-array
while (indexOfSubArrayOne < size1 && indexOfSubArrayTwo < size2) {
    if (tempArray1[indexOfSubArrayOne] < arr[ size1 + indexOfSubArrayTwo]) {
        arr[indexOfMergedArray] = tempArray1[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        } else {
        arr[indexOfMergedArray] = arr[ size1 + indexOfSubArrayTwo];
        indexOfSubArrayTwo++; }
    indexOfMergedArray++; }
while (indexOfSubArrayOne < size1) {
    arr[indexOfMergedArray] = tempArray1[indexOfSubArrayOne];
    indexOfSubArrayOne++; indexOfMergedArray++; }
while (indexOfSubArrayTwo < size2) {
    arr[indexOfMergedArray] = arr[size1 + indexOfSubArrayTwo];
    indexOfSubArrayTwo++; indexOfMergedArray++; }}
template <typename T>
struct SubarrayInfo {
    T* Arr = nullptr;
    int Size1;
    int Size2; };
void ConstructedSort(int* arr, int n) {
    const int minBlockSize = 32, maxBlockSize = 10000;
    std::stack<SubarrayInfo<int>> subarraysToMerge;
    std::stack<std::pair<int*, int>> unsortedArrays;
    int size1 = 0, size2 = 0, start = 1, end = 0, localSize = 0;
    bool swapped = false;
    int s = 0, e = n – 1, idx = 0;
    bool isSorted = false;
    auto pivotIdx = (s + e) / 2;
    int min, max;
```

Текст програми відповідний поточній хромосомі буде додано між початковою і кінцевою незмінними частинами. Далі представлено кінцеву незмінну частину, якій відповідає термінал $T_E$, у якій гарантовано відбувається злиття усіх відсортованих частин масиву і закриття останньої фігурної дужки функції.

```
// text ↲ T_E:
while(!subarraysToMerge.empty()) {
```

```
        const auto& stm =
subarraysToMerge.top();
        subarraysToMerge.pop();
        Merge(stm.Arr, stm.Size1, stm.Size2);
    }}
```

У результаті буде сформований готовий текст програми. Можливе як додавання файлів із сформованим кодом сортування до файлів програми, так і компіляція, і збірка окремої мінімалістичної бібліотеки.

Для перевірки спроможності адаптованих алгоритмів конкурувати з відомими алгоритмами сортування виконані експерименти із застосуванням запропонованого конструктивно-продукційного підходу на основі генетичного алгоритму.

## Експериментальні дослідження

Експерименти проводились над даними різної структури:

- повністю відсортовані дані;
- дані, відсортовані у зворотному порядку;
- повністю відсортований масив, у якому виконується декілька перестановок пар елементів, вибраних випадковим чином;
- відсоток невідсортованості. У повністю відсортованому масиві випадковим чином обирається певна кількість послідовних ділянок елеменів сумарною довжиною рівною даному відсотку довжини усього масиву. В межах кожної ділянки випадково обрані елементи міняються місцями певну кількість разів;
- повністю випадкові дані.

Проведені експерименти із вищезазначеними типами даних обсягом один мільйон і десять мільйонів елементів.

Для кожної популяції генетичного алгоритму генерувався 51 масив сортованих даних відповідної структури. Вони зберігались у 51 бінарний файл. Кожний індивід популяції, який представляє собою конкретний алгоритм сортування, вичитував масив даних із файла. Виконувалось сортування, перевірялось, чи сортування виконане успішно і чи зберігався результат – час виконання – у масив. Після сортування усіх масивів вхідних даних вираховувався медіанний час і зберігався у окремий бінарний файл. Таким чином використовувались ідентичні дані для кожного індивіда.

Під час формування наступної популяції, 20% хромосом переносились з попередньої, 40% - додавалися після схрещування, решта заповнювалась новими хромосомами.

Задля обмеження довжини хромосоми і, відповідно, довжини генерованої функції сортування, процес генерації хромосоми керувався за допомогою наступних параметрів. Параметр MGBS = 1 визначає мінімальну кількість генів, необхідну для того, щоб стали доступними для вибору гени розбиття, які додають нові вузли-нащадки до поточного вузла. Максимальна глибина дерева-хромосоми MD = 3. На початку генерації хромосоми для кореневого вузла глибина дорівнювала одиниці. При виборі гена розбиття масиву до поточного вузла додавались два вузли-нащадки із глибиною більшою, ніж у поточного вузла на одиницю. У разі глибини меншій, ніж максимальна, правило вибору гена фінального сортування недоступне. Натомість доступні правила вибору генів розбиття. При досягненні максимальної глибини замість гену розбиття масива підставляється ген фінального сортування і вузли нащадки не додаються. Таким чином будується збалансоване дерево-хромосома із заданою глибиною.

Результати експериментів наведені у табл. 2. У рядку із відповідним типом даних представлений кращий із загальновідомих алгоритмів і кращий із конструйованих (адаптованих).

Для повністю відсортованих даних покращення становить 35-37% порівняно з кращим із загальновідомих алгоритмів, а саме сортуванням вставками. Для даних відсортованих у зворотному порядку, конструйовані алгоритми показали найвищий рівень покращення – 92-94%. Це досягається наявністю гена SR на початку хромосоми.

Таблиця 2. Результати експериментів із сортування

| Обсяг сортованих даних | Тип вхідних даних | Алгоритм | Медіанний час, мкс | Покращення/Погіршення, % |
|---|---|---|---|---|
| Один мільйон | Повністю відсортовані | BSB_FSB_FSB_P_PUA_SEEI_MR_P_PUA_BGQP_BSB_BSF_FS_ESS_PUA_QP_FSB_BLQP_SEEI_MR_FS_ESS_ESS_PUA_BSF_BSF_QP_SEI_ML_SIS_PUA_FSB_FS_ESS_PUA_QP_SEI_ML_FS_ESS_ESS_ESS | 329 | 37.33% |
| | | InsertionSort | 525 | |
| | Відсортовані у зворотному порядку | SR_SEI_ML_FSS_BGPM_SIS_PUA_BSF_BSF_BLPM_FSB_P_PUA_SEI_ML_FSS_QP_FS_ESS_PUA_BSF_SEEI_MR_SEEI_MR_FS_ESS_ESS_PUA_BSF_QP_P_PUA_SEEI_SEEI_MR_BSF_SEI_ML_FS_ESS_PUA_FSB_SEI_ML_FS_ESS_ESS_ESS | 896 | 92.7% |
| | | QuickSort | 12276 | |
| | Декілька перестановок | InsertionSort | 2562 | -484% |
| | | P_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_ESS | 14985 | |
| | Відсоток невідсортованих елементів | P_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_ESS | 17847 | 3.41% |
| | | QuickSort | 18475 | |
| | Випадкові | P_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_ESS | 61831 | 0.24% |
| | | QuickSort | 61980 | |
| Десять мільйонів | Повністю відсортовані | BSF_SEEI_MR_SIS_PUA_SEEI_MR_BGQP_BSF_P_PUA_BSF_BSB_BSF_FSS_FS_ESS_PUA_SR_FS_ESS_ESS_PUA_SEEI_MR_PM_SIS_PUA_QP_FS_ESS_PUA_BLPM_SEEI_SEEI_MR_SR_FS_ESS_ESS_ESS | 3706 | 35.06% |
| | | InsertionSort | 5707 | |
| | Відсортовані у зворотному порядку | SR_BSF_P_PUA_FSB_SIS_PUA_FSS_BSB_FSB_QP_FS_ESS_PUA_FSB_SEI_SEEI_ML_MR_QP_FS_ESS_ESS_PUA_BSF_SR_P_PUA_SR_BSF_BLQP_FSS_FS_ESS_PUA_BSF_BSF_BSF_SEI_ML_FS_ESS_ESS_ESS | 8364 | 94.07% |
| | | QuickSort | 141151 | |
| | Декілька перестановок | P_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_ESS | 193621 | 0.137% |
| | | QuickSort | 193887 | |
| | Відсоток невідсортованих елементів | P_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_PUA_P_PUA_FS_ESS_PUA_FS_ESS_ESS_ESS | 201806 | 0.033% |
| | | QuickSort | 201872 | |
| | Випадкові | SIS_PUA_SIS_PUA_FS_ESS_PUA_FS_ESS_ESS_PUA_SIS_PUA_FS_ESS_PUA_FS_ESS_ESS_ESS | 715804 | 0.322% |
| | | QuickSort | 718110 | |

Сортування масивів із кількома перестановками обсягом один мільйон елементів показало значне погіршення – 484%. Для даних обсягом десять мільйонів наявне незначне покращення – 0.137%. Варто зазначити, що на масивах обсягом один мільйон сортування вставками показало значну ефективність, однак для десяти мільйонів елементів кращим із загальновідомих виявилось швидке сортування.

Для експериментальних даних із відсотком невідсортованих елементів покащення становить 0.33% (десять мільйонів) і 3.41% (один мільйон).

Покращення для сортування випадкових даних обсягом один мільйон становить 0.24%. Конструйований алгоритм включає гени розбиття із перестановкою елементів, що відповідають частині логіки швидкого сортування. Для даних обсягом десять мільйонів розраховане покращення 0.32%. Конструйонавий алгоритм включає гени розбиття, які відповідають частинам сортування злиттям.

## Висновки

У попередніх роботах була запропонована конструктивна модель хромосоми деревовидної структури яка кодує алгоритм сортування. Викладена спеціалізація та конкретизація конструктору формування хромосом.

У цій статті як логічне продовження попередньої роботи розроблено підхід до декодування хромосом у послідовність генів і їх подальшого перетворення на текст програми мовою програмування.

Застосування конструктивно-продукційного моделювання є ефективним підходом до побудови алгоритмів, здатних адаптивно формувати структуру рішення на основі чітко визначених продукційних правил та конструктивних принципів. Запропонований метод дав змогу системно описати процес створення алгоритмів як послідовність перетворень, у межах яких кожне правило робило внесок у формування цілісного алгоритму сортування.

Продемонстровано ефективність застосування генетичного алгоритму для структурної адаптації конструйованих алгоритмів сортування і вибору найбільш пристосованого до заданих умов використання. Проведені експерименти підтвердили, що еволюційний підхід здатний адаптивно вдосконалювати структуру алгоритму, поступово підвищуючи його продуктивність.

## Література


1. R. Sedgewick, ‘Implementing quicksort programs’, Communications of the ACM, 21(10), 847–857 (1978).
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. J. W. J. Williams, Algorithm 132 (heapsort), Communications of the ACM, 7, 347–348 (1964)
4. C. Song and H. Li, “Improvement of Counting Sorting Algorithm,” in Journal of Computer and Communications, vol. 11, pp. 12–22, 2023
5. J. Wassenberg and P. Sanders, “Engineering a multi-core radix sort,” in Proc. Euro-Par 2011, Part II, LNCS 6853, Bordeaux, France, 2011, pp. 160–169. doi: 10.1007/978-3-642-23397-5_16.
6. N. Faujdar and S. Saraswat, “The detailed experimental analysis of bucket sort,” in Proceedings of the 7th International Conference on Cloud Computing, Data Science & Engineering – Confluence, IEEE, pp. 12–13, Jan. 2017
7. B. Haeupler, R. Hladík, J. Iacono, V. Rozhoň, R. E. Tarjan and J. Tětek, “Fast and Simple Sorting Using Partial Information,” in Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 3953–3973, 2025, doi: 10.1137/1.9781611978322.134
8. M. Subramaniam, T. Tripathi and O. Chandraumakantham, “Cluster Sort: A Novel Hybrid Approach to Efficient In-Place Sorting Using Data Clustering,” in IEEE Access, vol. 13, pp. 74359–74374, 2025.
9. 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.
10. 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.
11. V.I. Shinkarenko, O.V. Makarov, Structural Adaptation of Sorting Algorithms Based on Constructive Fragments, CEUR Workshop Proceedings. – Vol. 3806. – pp. 16–29. – 2024

12. В.І. Шинкаренко, О.В. Макаров Конструктивно-продукційне моделювання хромосом генетичного алгоритму з закодованими алгоритмами сортування, Проблеми програмування, 2025, № 3, С. 39-52. doi: 10.15407/pp2025.03.039
13. D. Beasley, D. R. Bull and R. R. Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals," in University Computing, vol. 15, pp. 58–69, 1993
14. В.І. Шинкаренко, В.М. Ільман. Конструктивно-продукційні структури та їх граматичні інтерпретації. I. Узагальнена формальна конструктивно-продукційна структура . Кібернетика і системний аналіз. – 2014. – Т. 50, № 5. – С. 8–16

## References

1. R. Sedgewick, 'Implementing quicksort programs', Communications of the ACM, 21(10), 847–857 (1978).
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. J. W. J. Williams, Algorithm 132 (heapsort), Communications of the ACM, 7, 347–348 (1964).
4. C. Song and H. Li, "Improvement of Counting Sorting Algorithm," in Journal of Computer and Communications, vol. 11, pp. 12–22, 2023
5. J. Wassenberg and P. Sanders, "Engineering a multi-core radix sort," in Proc. Euro-Par 2011, Part II, LNCS 6853, Bordeaux, France, 2011, pp. 160–169. doi: 10.1007/978-3-642-23397-5_16.
6. N. Faujdar and S. Saraswat, "The detailed experimental analysis of bucket sort," in Proceedings of the 7th International Conference on Cloud Computing, Data Science & Engineering – Confluence, IEEE, pp. 12–13, Jan. 2017
7. B. Haeupler, R. Hladík, J. Iacono, V. Rozhoň, R. E. Tarjan and J. Tětek, "Fast and Simple Sorting Using Partial Information," in Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 3953–3973, 2025, doi: 10.1137/1.9781611978322.134
8. M. Subramaniam, T. Tripathi and O. Chandraumakantham, "Cluster Sort: A Novel Hybrid Approach to Efficient In-Place Sorting Using Data Clustering," in IEEE Access, vol. 13, pp. 74359–74374, 2025.
9. 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
10. 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
11. V.I. Shinkarenko, O.V. Makarov, Structural Adaptation of Sorting Algorithms Based on Constructive Fragments, CEUR Workshop Proceedings. – Vol. 3806. – pp. 16–29. – 2024
12. V. I. Shinkarenko, O. V. Makarov. Constructive-synthesizing modeling of the genetic algorithm chromosomes with encoded sorting algorithms, Problems of Programming. – 2025. – No. 3. – pp. 39–52. – DOI: 10.15407/pp2025.03.039
13. D. Beasley, D. R. Bull and R. R. Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals," in University Computing, vol. 15, pp. 58–69, 1993
14. 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

Дата першого надходження до видання: 09.03.2026
Внутрішня рецензія отримана: 13.03.2026
Зовнішня рецензія отримана: 14.03.2026
Дата прийняття статті до друку: 19.03.2026
Дата публікації: 16.04.2026

***Про авторів:***

*Шинкаренко Віктор Іванович,*
доктор технічних наук, професор
*Shynkarenko Victor,*
Ph.D (doctor), professor
https://orcid.org/0000-0001-8738-7225
E-mail: shinkarenko_vi@ua.fm

*Макаров Олексій Вікторович,* аспірант
*Makarov Olexiy,* Post-graduate student
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.
Ukrainian State University of Science and Technology (Dnipro)
E-mail: office@ust.edu.ua