УДК 004.4 http://doi.org/10.15407/pp2024.02-03.011

В.І. Шинкаренко, О.В. Макаров

ГЕНЕТИЧНИЙ АЛГОРИТМ ДЛЯ СТРУКТУРНОЇ АДАПТАЦІЇ АЛГОРИТМІВ СОРТУВАННЯ

Застосовано конструктивізм для формування коду алгоритму сортування. Представлено метаалгоритм генерації програмного коду. Для генерації використовуються частини існуючих алгоритмів сортування і допоміжні утиліти. Використано генетичний алгоритм для вибору алгоритму з максимальною часовою ефективністю у заданих умовах використання. Використання стандартного генетичного алгоритму стикається з проблемою, пов'язаною з різною кількістю елементарних дій по сортуванню, що призводить до використання хромосом різної довжини. Для рішення проблеми запропоновано представлення хромосоми у формі бінарного дерева. Кожен вузол має гени початку, кінця і двох вузлів-нащадків. Для формування алгоритму, який гарантовано буде сортувати масив, усі кінцеві вузли (листя) включають ген фінального сортування у кінець початкової послідовності генів. Даний ген декодується викликом існуючого алгоритму сортування, який гарантовано виконає сортування. Операції кросоверу та мутацій виконуються на хромосомах у формі бінарного дерева. Схрещення виконується за допомогою обміну вузлами дерева. Реалізовані механізми кодування і декодування алгоритму сортування із хромосоми. Для декодування і формування відповідного алгоритму сортування виконується лінеріалізація: формування текстового представлення за алгоритмом обходу дерева у глибину. Фітнес функція визначається як середній час сортування випадково сформованих масивів для сортування (для всіх хромосом однакові масиви) у деякому стабільному середовищі з урахуванням певних особливостей цих масивів. Передбачено застосування інших фітнес функцій пов'язаних з кількістю обчислень, порівнянь або перестановок. Розроблене програмне забезпечення має застосовуватись у процесі адаптації алгоритмів сортування до стабільних потоків вхідних даних та середовищ використання.

Ключові слова: алгоритм сортування, сортування, конструктивізм, генетичний алгоритм, хромосома, бінарне дерево.

V.I. Shinkarenko, O.V. Makarov

GENETIC ALGORITHM FOR STRUCTURAL ADAPTATION OF SORTING ALGORITHMS

Constructivism was applied to form the sorting algorithm code. The meta-algorithm of program code generation is presented. Parts of existing sorting algorithms and auxiliary utilities are used for generation. A genetic algorithm was used to select the algorithm with the maximum time efficiency under the given conditions of use. The use of a standard genetic algorithm faces a problem associated with a different number of elementary sorting operations, which leads to the use of chromosomes of different lengths. To solve the problem, a representation of the chromosome in the form of a binary tree is proposed. Each node has start genes, end genes, and two child nodes. To form an algorithm that is guaranteed to sort the array, all end nodes (leaves) include the final sorting gene at the end of the start genes sequence. This gene is decoded by calling the existing sorting algorithm, which is guaranteed to perform sorting. Crossover and mutation operations are performed on chromosomes in the form of a binary tree. Crossover is performed using the exchange of tree branches. Mechanisms of coding and decoding of the sorting algorithm from chromosome have been implemented. Decoding and formation of a suitable sorting algorithm is performed using linearization: formation of a textual representation using a depth-first tree traversal algorithm. The fitness function is defined as the average time of sorting randomly generated arrays for sorting (identical arrays for all chromosomes) in some stable environment, considering certain features of these arrays. It is possible to use other fitness functions related to the number of calculations, comparisons, or permutations. The developed software should be used for adaptation of sorting algorithms to stable input data streams and environments.

Key words: sorting algorithm, sorting, constructivism, genetic algorithm, chromosome, binary tree.

Вступ

Із появою Інтернету і цифрової технології взагалі, стрімко зростають обсяги даних, що генеруються та зберігаються. З цифровізацією більшості сфер життя – від бізнесу та науки до особистих комунікацій – виникає потреба в ефективному управлінні цими даними. Зростання обсягів даних ставить перед нами виклики, пов'язані зі збереженням, обробкою, аналізом та інтерпретацією.

У такому контексті актуальність ефективних алгоритмів сортування стає критичною. Для ефективної роботи з великими обсягами даних необхідно швидко та ефективно впоратися з їх обробкою. Навіть найпростіші операції, такі як сортування, можуть стати часо- та ресурсо- затратними, якщо використовуються неефективні методи.

Ефективні алгоритми сортування дозволяють швидко опрацьовувати великі обсяги даних, що є критичним для багатьох застосувань. У деяких мережевих пристроях сортування даних може використовуватися для оптимізації обробки пакетів даних та керування мережевим трафіком. Системи керування базами даних (СКБД) використовують алгоритми сортування для виконання запитів, об'єднання даних та інших операцій [1]. У розподілених системах зберігання даних, таких як Hadoop або Apache Spark [2], алгоритми сортування використовуються для обробки великих обсягів інформації та забезпечення швидкодії.

Еволюція алгоритмів сортування є захоплюючим шляхом в історії обчислювальної науки, який почався з простих, але ефективних методів. Наприклад, сортування бульбашкою (Bubble sort) або вставками (Insertion sort) із обчислювальною складністю $O(n^2)$ [3]. У подальшому з'явились більш складні та оптимізовані алгоритми, більш придатні для сортування великих обсягів даних. Такі як швидке сортування (Quick sort) чи сортування злиттям (Merge sort), із обчислювальною складністю у середньому $O(n * log(n))$. У подальшому зростання вимог до стабільності і швидкості сортування привели до появи комбінованих алгоритмів. Найвідомішими представниками яких є Timsort [4] – симбіоз сортувань вставками та злиттям. Інтроспективне сортування (Introsort) [5] починає із швидкого сортування, потім за певних умов переходить на пірамідальне сортування (Heapsort) і викликає сортування вставками для невеликих послідовностей.

Комбіновані алгоритми поєднують у собі переваги складових для підвищення ефективності. У [6] розглядається новітній підхід до формування, перетворення та аналізу конструкцій за допомогою операцій зв'язування, підстановки, виводу тощо.

Для формування структур (складових та їх взаємного розташування) алгоритмів сортування застосований підхід конструктивно-продукційного моделювання. Це забезпечує вирішення наступних задач:

- створення нових алгоритмів сортування із частин існуючих;
- адаптація алгоритмів сортування до сортованих даних;
- адаптація структур даних в оперативній пам'яті.

В даній роботі не розглядається теоретична модель, а лише за цією моделлю її практичне застосування.

Мета

Метою даної роботи є розробка генетичного алгоритму [7] для структурної адаптації алгоритмів сортування. Структурна адаптація алгоритмів сортування полягає у формуванні адаптованого алгоритму з частин відомих алгоритмів таким чином, щоб він був не гіршим за часовими показниками від інших алгоритмів сортування в деякому стабільному середовищі використання.

Особливістю генетичного алгоритму для даної задачі є формування хромосом невизначеної довжини і складу з можливістю як кодування, так і декодування у алгоритм сортування.

Конструювання алгоритмів сортування

Для конструювання алгоритмів сортування будемо використовувати частини існуючих загальновідомих алгоритмів сортування. Використовувались у поточній версії програми такі базові (атомарні) операції з даних алгоритмів:

- швидке сортування (Quick sort). Розділення масиву даних на дві частини відносно обраного елемента, який називається опорним (pivot). Усі елементи, менші за опорний, переставляються ліворуч від нього, а всі більші елементи – праворуч;
- сортування вставками (Insertion sort). Вставка одного елемента із невідсортованої частини масиву у відсортований підмасив;
- сортування вибором (Selection sort). Пошук мінімального або максимального елемента у невідсортованому підмасиві і перестановка;
- шейкерне сортування (Cocktail shaker sort). Прохід масивом у прямому або зворотному напрямку і перестановка усіх пар елементів, що стоять у зворотному порядку;
- сортування злиттям (Merge sort). Злиття двох відсортованих масивів у один.

Також можливою операцією є умовне розбиття масиву на дві частини і сортування кожної окремо. Після того, як кожна частина буде відсортована необхідно виконати злиття у єдиний відсортований масив.

Модель передбачає додавання інших базових операцій. Це можуть бути частини існуючих алгоритмів сортування. Наприклад, вставка елемента із певним кроком, використана у сортуванні Шелла. Також доцільне використання детермінованих і стохастичних передобробок [6, 8], або

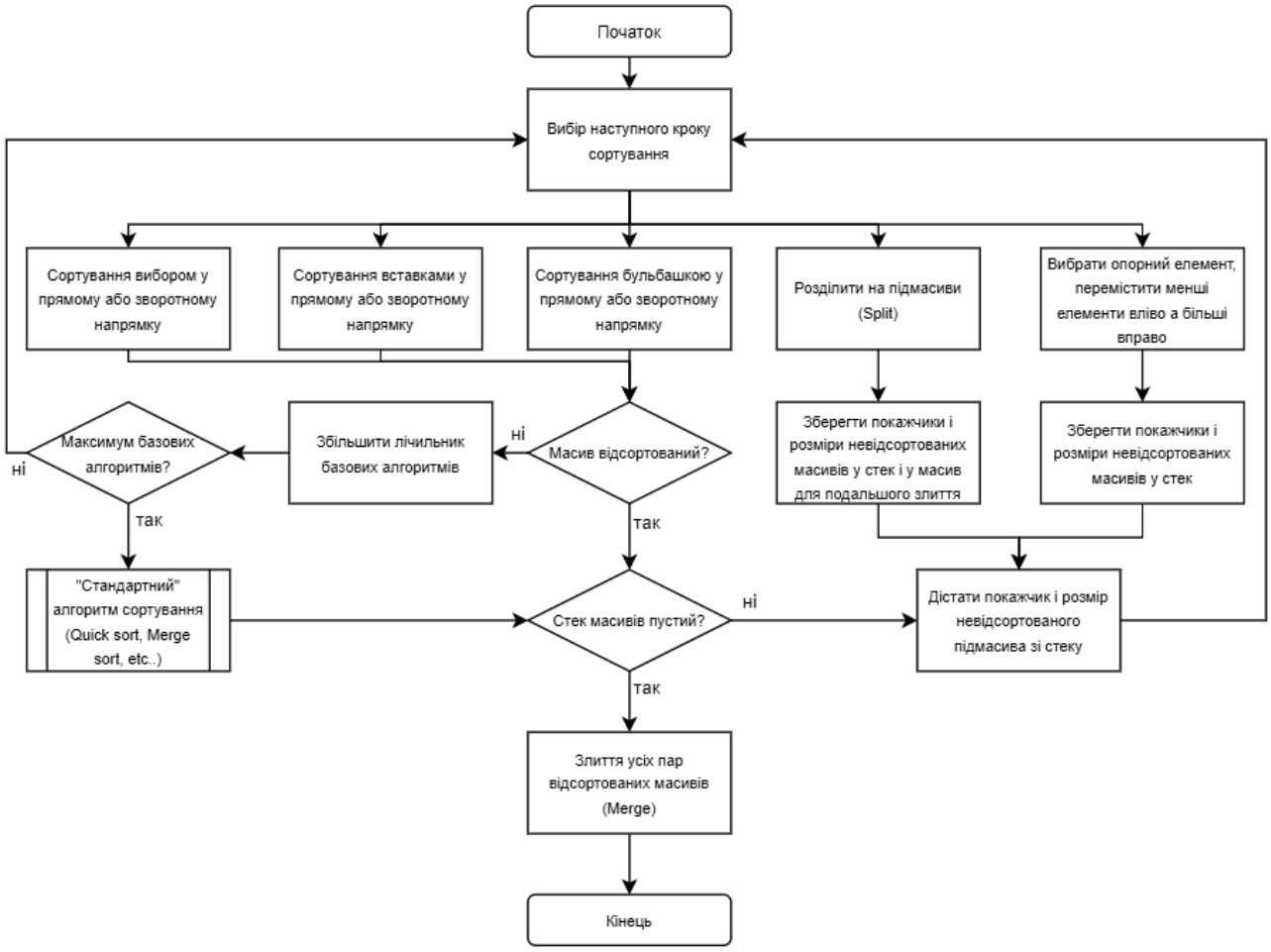

Рис. 1. Блок-схема метаалгоритму генерації програмного коду

їхніх складових операцій. Це може підвищити ефективність кінцевого алгоритму.

Правила формування алгоритму. Для формування алгоритму (рис. 1) складові базові операції вибираються випадково. Таким чином можливі будь-які комбінації атомарних алгоритмів і, як результат, безліч унікальних конструйованих алгоритмів сортування.

Деякі базові алгоритми мають спеціальні вимоги до їхнього використання. У разі, якщо вимоги не виконуються, генерація буде неможливою або конструйований алгоритм не виконуватиме повне сортування масиву. Наприклад, під час використання розбиття масиву на дві частині і сортування кожної частини окремо, має викликатись операція злиття в один відсортований масив.

Розглянемо проблеми з комбінацією алгоритмів сортування. Атомарні частини алгоритмів сортування вибором і бульбашкою розділяють масив на невідсортовану (або ще не проаналізовану) частину та відсортовану. Якщо виконуються проходи у прямому та зворотному напрямку, то масив ділиться на дві відсортовані частини на початку і в кінці масиву та невідсортовану. Через особливості логіки алгоритмів відсортовані частини мають у собі елементи строго менші (на початку) або більші (у кінці масиву), ніж ті, що лишаються у невідсортованій серединній частині. Якщо використовувати проходи тільки в один бік, то отримаємо класичне сортування вибором або бульбашкою. У цьому випадку масив буде повністю відсортований. Якщо використовувати проходи у різних напрямках, то відсортовані частини на початку і в кінці будуть збільшуватись доки не зустрінуться і утворять єдиний відсортований масив.

Для класичного сортування вставками – коли усі вставки виконуються тільки у ліву (або тільки у праву) частину – також результатом роботи буде повністю відсортований масив. Але, якщо виконувати вставки у початок і кінець масиву, то будемо отримувати відповідно два відсортованих масиви. Але немає жодних гарантій, що елементи правого масиву строго більші або рівні елементам з лівого. Для об'єднання двох відсортованих підмасивів у один необхідно викликати операцію злиття (Merge).

Додаткова складність виникає у процесі комбінування алгоритмів, описаних вище. Припустимо, що сортується масив довжиною N. Після виконання M операцій вибору в початок або проходів сортуванням бульбашкою у зворотному напрямку гарантовано матимемо відсортований підмасив [0, M-1] зліва, усі елементи якого строго менші або рівні решті елементів у невідсортованій частині. Якщо тепер виконати вставку елемента під індексом M у ліву частину, то матимемо відсортований масив [0, M], але тепер його елементи не будуть менші, ніж ті, що лишились у невідсортованій частині.

Подальші виклики операцій вибору можуть бути не ефективні через те, що найменший елемент із невідсортованої частини масиву може виявитись меншим, ніж той, що додався сортуванням вставкою. І ліва відсортована частина після додавання цього елемента стає невідсортованою, що означає неефективність створеного алгоритму.

Для сортування бульбашкою одна операція вставки не є критичною. Якщо елемент доданий вставкою під індексом M більший, ніж наступний (M+1), що буде доданий сортуванням бульбашкою, то на першому кроці вони поміняються місцями і підмасив буде відсортований. Однак, якщо була виконана вставка і хоча б одна операція вибору, то сортування бульбашкою втрачає ефективність, і масив може лишитись у невідсортованому стані (рис. 2).

2	7	4	3	1	0	Бульбашкою
0	**2**	7	4	3	1	Вибором
0	1	**7**	4	3	2	Вставкою
0	1	7	**4**	3	2	Вибором
0	1	7	2	**3**	4	**Ліва частина невідсортована**

Рис. 2. Приклад невірної послідовності базових алгоритмів

Для вирішення вищезгаданої проблеми введемо нові атомарні операції – злиття зліва (ML) і злиття справа (MR). Для відстежування викликів операцій вставки будуть використовуватись додаткові індикатори. Якщо було використано операцію вставки у початок, відповідний індикатор буде встановлено. Перед наступним викликом операцій сортування бульбашкою або вибором для поточного масиву, потрібно буде спочатку викликати операцію злиття зліва. Тобто зберегти покажчики відсортованої частини зліва і решти масиву для подальшого злиття. Для аналогічної ситуації у кінці масиву буде викликатись операція злиття справа.

Застосування генетичного алгоритму для вибору найбільш ефективного алгоритму сортування

Хромосому представимо послідовністю генів – базових алгоритмів сортування та допоміжних утиліт. Для представлення хромосоми у текстовому вигляді задаємо текстове представлення кожному гену. Таким чином хромосома буде мати послідовність генів, розділених символом - «,». Через те, що кожен ген являє собою якусь функцію, то використаємо для текстового представлення її абревіатуру. Розглянемо існуючі гени:

- BSB (Bubble sort backward) – один прохід масивом із кінця у початок із перестановкою пар елементів, що йдуть у зворотному порядку;
- BSF (Bubble sort forward) – те саме, що і BSF, тільки прохід від початку у кінець масиву;
- FSB (Find swap biggest element) – пошук найбільшого елемента у невідсортованій частині масиву і перестановка на поточне місце;
- FSS (Find swap smallest element) – те саме що і FSB, тільки виконується пошук найменшого елемента;
- SEEI (Single element end insertion) – вставка поточного елемента у відсортовану частину в кінці масиву;
- SEI (Single element insertion) – вставка поточного елемента у відсортовану частину на початку масиву;
- SIS (Split in subarrays) – 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;
- FS (Final sort) – один із загальновідомих алгоритмів сортування, який гарантовано виконає сортування;
- PUA (Pop unsorted array) – дістати покажчик і розмір наступної невідсортованої частини для сортування. Виконується для кожної частини збереженої операцією SIS;
- ESS (End subarray sort) – допоміжна операція, яка закриває фігурні дужки, відкриті попередніми операціями для перевірки індикатора відсортованості. Виконується для кожної операції PUA після послідовності генів сортування;
- ML (Merge left) – збереження покажчиків на відсортовану частину на початку та решту масиву для подальшого злиття в один відсортований масив;
- MR (Merge right) – те ж саме, що і ML тільки для відсортованої частини в кінці масиву;

Приклад сформованої хромосоми у текстовому вигляді:
SEEI, MR, FSB, SIS, PUA, BSF, FSS, FS, ESS, PUA, SEI, ML, FSS, FS, ESS, ESS.

Для обмеження довжини хромосоми, введемо певні обмеження.

На початку сортування вхідного масиву глибина дорівнює одному. У процесі розбиття масиву і переходу до сортування будь-якої із його частин, глибина збільшується на одиницю. Таким способом обмежується кількість розбивань масиву на частини.

Окремим параметром обмежується кількість базових операцій сортування для кожного підмасиву. При досягненні максимального значення, викликається розбиття або фінальне сортування.

Генетичний алгоритм для генерації алгоритмів сортування. Кожен індивідуум представляє собою алгоритм сорту-

вання. Генами представлені атомарні операції-складові існуючих алгоритмів сортування та допоміжні утиліти.

Функцією придатності, яка оцінює якість кожного індивідуума, буде часова ефективність. Але можуть бути і такі: кількість порівнянь, кількість перестановок тощо. Або зважена комбінація таких чинників.

Генерація наступної популяції здійснюється за допомогою схрещування і мутацій. Схрещування здійснює обмін частинами між двома батьками для створення нового індивідуума.

Мутація випадковим чином змінює деякі елементи генетичної послідовності. Заново генеруються деякі ділянки хромосоми і випадково вибираються атомарні операції.

Для наступного покоління обираються індивідууми на основі їхньої придатності. Індивіди з більшою придатністю мають більше шансів вижити і брати участь у створенні нових індивідів.

Певна кількість індивідів з найкращими показниками переноситься в наступне покоління без змін. Інші формуються схрещуванням індивідів існуючої популяції. Індивіди для схрещення можуть вибиратись випадково, або за певними правилами. Також можливий варіант із схрещенням кращих індивідів за правилами, а решти – випадково. Для урізноманітнення популяції додається певна кількість індивідів, згенерованих випадковим чином, так само, як була створена найперша популяція.

Встановлюються параметри генетичного алгоритму, такі як розмір популяції, ймовірність мутації, кількість поколінь тощо. Визначається умова зупинки, наприклад, максимальна кількість поколінь або досягнення певного рівня придатності.

Проводиться декілька ітерацій генетичного пошуку, оптимізуються параметри і функція придатності для покращення результатів.

Застосування генетичного алгоритму до генерації алгоритмів сортування дозволяє автоматично еволюціонувати рішення, призначені для різних сценаріїв, та адаптувати їх до умов, що постійно змінюються.

Представлення хромосоми у вигляді дерева

Вхідний масив може мати будь-який відсоток відсортованості. Як тільки дані у масиві будуть відсортовані, доцільно завершити виконання для уникнення погіршення часової ефективності. Відстежування такої ситуації реалізуємо використанням ознаки відсортованості масиву, яка буде перевірятись після кожної базової операції сортування.

Постійне оновлення і перевірки ознаки відсортованості масиву вводить додаткові правила та обмеження у процес генерації хромосоми. У коді кожна наступна перевірка додає новий рівень вкладеності і область видимості, обмежену фігурними дужками “{“ та “}”. Перевірка ознаки відсортованості і відкриття області видимості має у собі кожний ген, що представляє базову операцію сортування. Відповідно кінцева ділянка хромосоми повинна мати фігурні дужки, що закриваються у кількості рівній відкритим дужкам. При розбитті масиву на підмасиви і сортуванні кожного окремо, матимемо послідовність базових операцій сортування і відповідну кількість закритих дужок. Оптимальною структурою даних для представлення вище описаного підходу є бінарне дерево (рис. 3).

Кожен вузол представляє собою частину хромосоми (послідовність генів) що відповідає сортуванню певної частини масиву. Окремо зберігаються гени початку і кінця. Вузли нащадків створюються під час розбиття масиву і сортування частин масиву окремо. Наприклад, під час розбиття масиву на дві рівні частини створюються два вузла нащадків. Кожний створений вузол матиме частину хромосоми, що реалізує сортування відповідної частини масиву. Масив генів кінця включатиме у собі злиття відсортованих масивів і ген завершення сортування поточного масиву, що включає закриття фігурних дужок і обнуління змінних.

Генерація алгоритму із хромосоми повинна відбуватись за принципом обходу

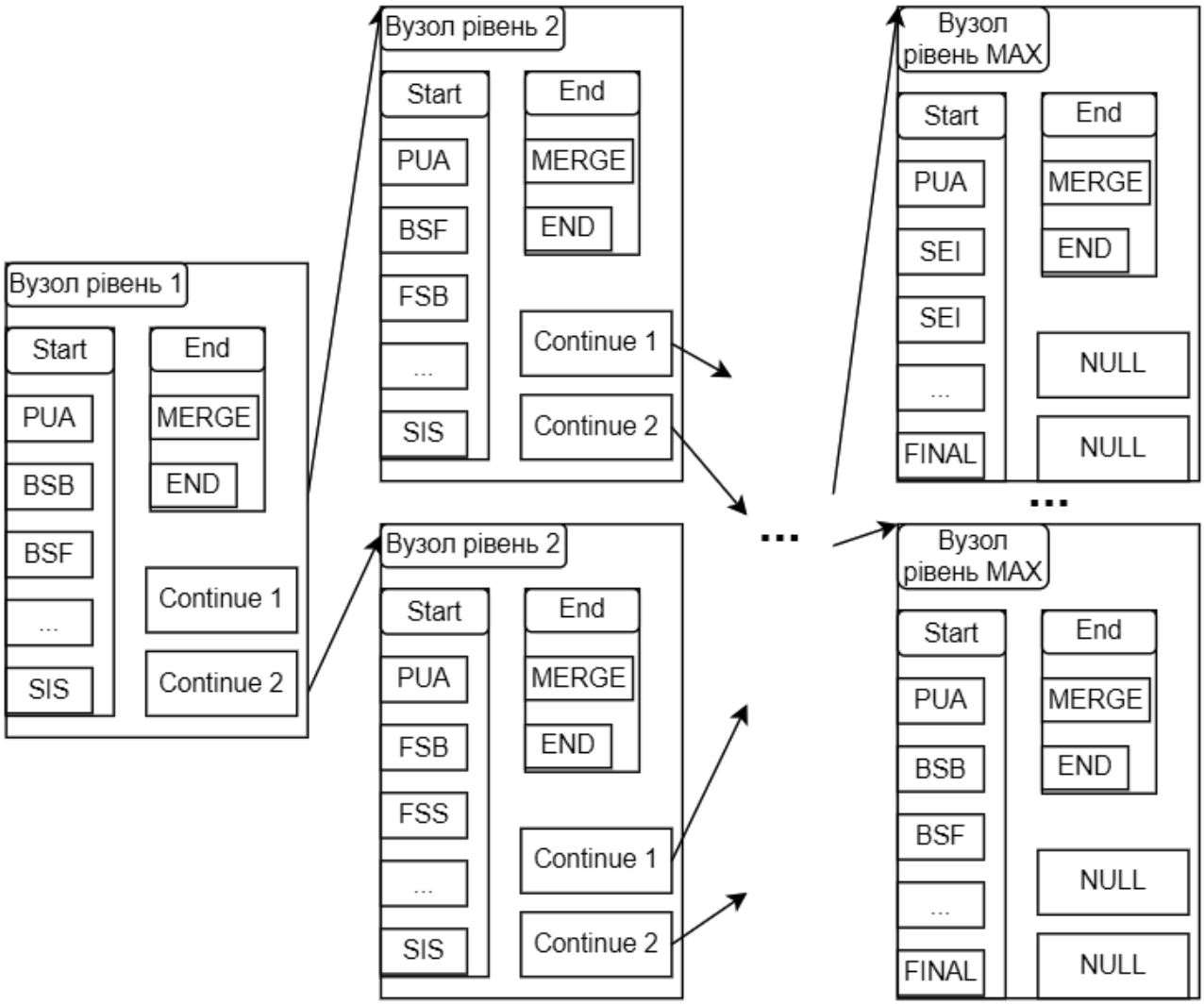

Рис. 3. Схема хромосоми у формі дерева

дерева вглиб [9]. Тобто для кожного вузла гени ідуть у наступній послідовності:

- гени початку;
- гени першого дитячого вузла;
- гени другого дитячого вузла;
- гени кінця.

Операція схрещення деревовидних хромосом виконується як обмін вузлами. Для збереження глибини дерева-хромосоми обмін може бути проведений тільки між вузлами відповідного рівня. Однак через те, що вершини на останньому рівні мають ген фінального сортування, а це гарантовано відсортує масив, обмін можливо проводити між вузлами різних рівнів.

Висновки

Розроблений генетичний алгоритм для структурної адаптації алгоритмів сортування. Його застосування дозволяє здійснити структурну адаптацію і вибрати найбільш ефективні та пристосовані до стабільних середовищ використання.

Представлений підхід до формування хромосоми у вигляді дерева дозволяє вирішити наявні проблеми. Він дозволяє ефективно формувати хромосоми, результатом декодування яких буде алгоритм, який гарантовано виконує сортування.

Формалізована модель формування алгоритмів сортування засобами конструктивно-продукційного моделювання є до-

сить об'ємною та буде представлена окремою статтею.

Розроблене програмне забезпечення має застосовуватись у процесі адаптації алгоритмів сортування до стабільних потоків вхідних даних та середовищ використання.

Література

1. А.Ю. Берко, О.М. Верес, В.В. Пасічник, Системи баз даних та знань. Книга 2, Магнолія-2006, 2013.
2. O. Mendelevitch, C. Stella, D. Eadline, Practical Data Science with Hadoop and Spark, Addison-Wesley, 2016.
3. D. Knuth, The Art Of Computer Programming, vol. 3: Sorting And Searching, Addison-Wesley, 1973.
4. Timsort. Accessed: 08.07.2023. https://svn.python.org/projects/python/trunk/Objects/listsort.txt.
5. D.R. Musser, Introspective Sorting and Selection Algorithms, *Software: Practice and Experience*, 1997, № 27 (8), pp.983–993. doi: 10.5555/261387.261395.
6. 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.
7. J.H. Holland, Adaptation in Natural and Artificial Systems, The MIT Press, 1992.
8. 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.
9. R. Tarjan, Depth-First Search and Linear Graph Algorithms, *SIAM Journal on Computing*, 1972, № 1(2). doi: 10.1137/0201010.

References

1. A.Yu. Berko, O.M. Veres, V.V. Pasichnik, Database and knowledge systems. Book 2, Magnolia-2006, 2013.
2. O. Mendelevitch, C. Stella, D. Eadline, Practical Data Science with Hadoop and Spark, Addison-Wesley, 2016.
3. D. Knuth, The Art Of Computer Programming, vol. 3: Sorting And Searching, Addison-Wesley, 1973.
4. Timsort. Accessed: 08.07.2023. https://svn.python.org/projects/python/trunk/Objects/listsort.txt.
5. D.R. Musser, Introspective Sorting and Selection Algorithms, in: Software: Practice and Experience, 1997, 27 (8), pp.983–993. doi: 10.5555/261387.261395.
6. V.I. Shinkarenko, A.Yu. Doroshenko, O.A. Yatsenko, V.V. Raznosilin, K.K. Galanin, Bicomponent sorting algorithms, in: Problems of programming, 2022, № 3-4, pp.32-41. doi: 10.15407/pp2022.03-04.032. [in Ukrainian]
7. J.H. Holland, Adaptation in Natural and Artificial Systems, The MIT Press, 1992.
8. V.I. Shinkarenko, O.V. Makarov, Study of the effectiveness of some deterministic preprocessing methods of data sorting, in: Problems of programming, 2023, № 4, pp.3-14. doi: 10.15407/pp2023.04.003. [in Ukrainian]
9. R. Tarjan, Depth-First Search and Linear Graph Algorithms, in: SIAM Journal on Computing, 1972, № 1(2). doi: 10.1137/0201010.

Одержано: 14.04.2024
Внутрішня рецензія отримана: 26.04.2024
Зовнішня рецензія отримана: 29.04.2024

Про авторів:

Шинкаренко Віктор Іванович,
доктор технічних наук, професор,
https://orcid.org/0000-0001-8738-7225

Макаров Олексій Вікторович,
аспірант,
https://orcid.org/0009-0003-0921-155X

Місце роботи авторів:

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