Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 1 (35) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 1 (35) *ISSN 2524-2296 (online)*

УДК 004.415.2:004.415.53 DOI: https://doi.org/10.30837/2522-9818.2026.1.039

Коломійцев О. В., Гулевич М. В., Дмітрієв О. М., Левченко А. О., Балабуха О. С.

ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ ВДОСКОНАЛЕНИХ ЖАДІБНОГО Й ДЕЛЬТА-ДЕБАГІНГ АЛГОРИТМІВ ОПТИМІЗАЦІЇ ТЕСТОВИХ СЦЕНАРІЇВ ДЛЯ C++ БІБЛІОТЕК

Ефективна оптимізація тестових сценаріїв (ТС) є необхідною умовою для підвищення ефективності тестування C++ бібліотек. **Предметом дослідження** є алгоритми оптимізації ТС для C++ бібліотек. **Мета роботи –** підвищення ефективності тестування C++ бібліотек завдяки вдосконаленню класичного алгоритму жадібного формування (оптимізації) ТС і алгоритму дельта-дебагінг мінімізації ТС для C++ бібліотек. **Завдання.** Удосконалити жадібний алгоритм оптимізації ТС і усунути його детермінованість, що забезпечить ефективне стиснення ТС зі збереження гілкового покриття коду. Удосконалити алгоритм дельта-дебагінг мінімізації тестових сценаріїв для ТС із розрідженим розташуванням надлишкових дій. Оцінити ефективності вдосконалених алгоритмів оптимізації ТС для C++ бібліотек порівняно з класичними алгоритмами. У роботі застосовано такі **методи**: жадібний алгоритм оптимізації тестових сценаріїв, алгоритм дельта-дебагінг мінімізації ТС, математичне моделювання та методи статистичного аналізу. Ефективність удосконалених алгоритмів оцінено на основі статистичного аналізу 100 моделювань для кожного алгоритму на двох C++ бібліотеках із відкритим вихідним кодом різної структурної складності. **Результати** оцінювання ефективності вказують на те, що вдосконалені алгоритми забезпечують збереження (збільшення) гілкового покриття з коефіцієнтом до 1,058, підвищення коефіцієнта стиснення ТС до 0,86 та скорочення часу на виконання тестового набору (ТН) від 1,5 до 2,5 раза, якщо порівнювати з класичними алгоритмами. **Висновки.** Удосконалені алгоритми істотно зменшують час на тестування C++ бібліотек без втрати гілкового покриття коду. Запропоновано вдосконалений жадібний алгоритм оптимізації тестових сценаріїв, у якому вибір дій ТС здійснюється з огляду на їх майбутню користь, визначену за очікуваним приростом гілкового покриття коду. Такий підхід усуває детермінованість класичного жадібного алгоритму, підвищує інформативність тестових сценаріїв і забезпечує збалансованість між збереженням гілкового покриття та стисненням ТС. Запропоновано вдосконалений алгоритм дельта-дебагінгу, який виконує групове вилучення неунікальних дій ТС за внеском у гілкове покриття, що дає змогу суттєво знизити довжину тестових сценаріїв без втрати ефективності тестування.

Ключові слова: програмне забезпечення; тестовий сценарій; алгоритм; оптимізація; покриття; часові витрати; дефект.

1. Вступ

Якість програмного забезпечення (ПЗ) неможливо підтримувати без ефективного тестування. Проте в процесі розвитку й супроводження проєктів тестові набори (ТН) швидко зростають, накопичуючи надмірність. Така надмірність призводить до збільшення часу на їх виконання, підвищення обчислювальних витрат і ускладнення їх підтримки. Тому завдання оптимізації та мінімізації тестових сценаріїв (ТС) залишається ключовою з практичного й наукового погляду.

У роботі [1] запропоновано інструмент CIDER, призначений для автоматизації тестування C++ бібліотек. Аналіз методів автоматизованої генерації ТС [2] демонструє переваги еволюційних, стохастичних і навчальних методик оптимізації тестових сценаріїв.

Класичними підходами щодо оптимізації ТН є жадібні й дельта-дебагінг алгоритми. Жадібні алгоритми основані на покроковому відборі ТС, що забезпечують найбільший внесок у покриття й використовуються як основна лінія для оцінювання сучасних методів і алгоритмів. Дельта-дебагінг алгоритм дає змогу скорочувати ТС завдяки поетапному вилученню частин ТС із перевіркою збереження необхідної поведінки. Крім стандартної версії, застосовуються варіації, які вилучають одразу групи тестових сценаріїв, що пришвидшує мінімізацію. Однак наявні варіанти залишаються схильними до детермінованості й не беруть до уваги динамічні властивості ТС.

Отже, удосконалення класичних жадібного й дельта-дебагінг алгоритмів оптимізації ТС для C++ бібліотек і оцінювання їх ефективності є актуальним науковим завданням.

2. Аналіз літературних джерел і визначення проблеми

2.1. Класичні методи оптимізації ТН

Задача мінімізації ТС (*Test Suite Minimization,* TSM) є однією з основних у галузі програмного тестування.

© Коломійцев О.В., Гулевич М.В., Дмітрієв О.М., Левченко А.О., Балабуха О.С., 2026

Вона полягає в пошуку піднабору тестових сценаріїв, який зберігає необхідне покриття або здатність виявляти дефекти ПЗ, проте водночас має менший розмір. Доведено, що задача TSM тісно пов’язана із задачею покриття множин і є NP-складною [3], що привело до появи різних евристичних і наближених алгоритмів.

Важливий систематичний огляд запропоновано в роботі [4], де узагальнено понад 113 досліджень, присвячених технікам скорочення ТН (*Test Suite Reduction*, TSR). Порівняно ефективність жадібних, кластеризаційних і еволюційних методів. Оцінено якість проведених досліджень. Запропоновано практичні рекомендації щодо відтворюваності результатів. Визначено стандарти оцінювання ефективності алгоритмів мінімізації, що застосовуються в більшості наукових робіт.

У систематичному огляді [5] проведено класифікацію методів регресійного тестування, зокрема з методами щодо селекції, пріоритизації та мінімізації ТН. Наголошено, що TSM має особливе значення у великих системах, де тестові набори можуть містити тисячі ТС.

У праці [6] емпірично оцінено техніки TSM, де продемонстровано, що прості евристичні алгоритми забезпечують скорочення розміру ТН на понад 70% у середньому. Спостерігається зниження здатності виявляти помилки приблизно на 12,5%, що обмежує практичну застосовність досліджених методів.

У роботі [7] окреслено використання еволюційних і локальних алгоритмів для скорочення ТН. Водночас продемонстровано, що, хоча еволюційні методи, зокрема генетичні алгоритми, демонструють високу ефективність, жадібні підходи також мають конкурентні результати в мультимодальному пошуковому просторі.

У дослідженні [8] запропоновано кластеризаційний метод щодо пріоритизації ТС, який формує групи сценаріїв за їх динамічною поведінкою під час виконання. Метод дає змогу скоротити кількість попарних порівнянь і підвищує ефективність виявлення збоїв, якщо порівнювати з методами, орієнтованими на збереження покриття.

У статті [9] продемонстровано можливість інтеграції пошукових алгоритмів для промислових продуктових ліній, доведено, що поєднання відбору, пріоритизації та мінімізації забезпечує найкращий баланс між часом на виконання й рівнем покриття.

Розглянуті дослідження заклали концептуальне підґрунтя сучасних методів TSM, визначивши оптимізацію покриття як багатокритеріальну задачу, у якій компроміс між точністю та швидкодією є ключовим чинником якості алгоритму.

2.2. Жадібні алгоритми оптимізації ТН

Жадібні алгоритми (*Greedy algorithms*) залишаються найпоширенішим інструментом для скорочення ТН. Основна ідея полягає в покроковому виборі ТС, які дають найбільший приріст критерію ефективності серед невідвіданих елементів.

У роботі [10] використано багаторівневу цільову функцію, яка дає змогу балансувати між різними цілями мінімізації, зокрема збереження (збільшення) покриття й усунення надмірності.

У статті [11] жадібні алгоритми оцінювалися за допомогою мутаційного тестування з огляду на кількість дефектів, що були виявлені сформованим тестовим набором. Ефективність жадібних алгоритмів підтверджено для регресійного тестування, де їх застосування забезпечує суттєве скорочення розміру ТН і часу на виявлення дефектів.

Зокрема результати в роботі [12] вказують на скорочення ТН на понад 75%, що демонструє практичну придатність жадібної пріоритизації для промислових програмних проєктів.

Крім того, у смарт-системах запропоновано шаблонно орієнтоване скорочення ТН для смарт-контрактів [13].

У дослідженні [14] проведено комплексне емпіричне оцінювання жадібних алгоритмів TSM за умови різних рівнів їх структурної складності. Доведено, що ефективність таких методів суттєво залежить від співвідношення кількості ТС і тестових вимог.

Систематичний огляд [15] підтверджує значущість жадібних алгоритмів порівняно з іншими евристичними й пошуковими методами.

Подальші дослідження зосереджені на вдосконаленні жадібних алгоритмів. Зокрема запропоновано прискорений варіант жадібного алгоритму (*Accelerated Greedy Additional Algorithm*, AGA), який підвищує ефективність пріоритизації ТН [16].

Більш нові роботи поєднують жадібні алгоритми з алгоритмами машинного навчання, наприклад, у скороченні ТН на основі подібності [17] або застосуванні механізмів часткової уваги для покращення відбору ТС [18].

Отже, жадібні алгоритми формують основу багатьох сучасних досліджень оптимізації ТН, і водночас є ключовими евристиками або складниками гібридних

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 1 (35) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 1 (35) *ISSN 2524-2296 (online)*

методів, які інтегрують додаткові критерії покриття, подібності або ефективності. Роль у літературі засвідчує як стабільність результатів, так і гнучкість для адаптації в нових доменах тестування ПЗ.

Отже, жадібні алгоритми є надійною основою для методів оптимізації ТС і залишають простір для вдосконалень.

2.3. Дельта-дебагінг алгоритми мінімізації ТН

Алгоритм дельта-дебагінгу (*Delta Debugging*) вперше запропонований *Zeller* і *Hildebrandt* [19] для спрощення й локалізації вхідних даних, які викликають програмні збої. Згодом його почали застосовувати для мінімізації ТС. Класична версія алгоритму працює за принципом послідовного вилучення частин тестових сценаріїв із перевіркою щодо збереження необхідної поведінки.

У дослідженні [20] розширено цей алгоритм. Застосовано дельта-дебагінг для локалізації причин збоїв у складних програмних системах, що дає змогу визначати мінімальні підмножини інструкцій, відповідальних за помилку. Такий підхід є відправною точкою для напряму локалізації дефектів (*Fault Localization*, FL).

У статті [21] продемонстровано, що техніка програмного зрізання (*Program Slicing*, PS) має спільні з дельта-дебагінгом принципи редукції й може використовуватись для мінімізації ТС без втрати релевантних способів виконання. Такий зв'язок між програмним зрізанням і дельта-дебагінгом задав основу для гібридних методів редукції в сучасних системах тестування.

Класичний алгоритм дельта-дебагінгу набув подальшого розвитку в методі ієрархічного дельта-дебагінгу (*Hierarchical Delta Debugging*, HDD), який бере до уваги структурованість вхідних даних і застосовує послідовне спрощення на рівнях їх ієрархії. Такий підхід забезпечує значне прискорення мінімізації та покращує якість отриманих скорочених вхідних даних порівняно з основним алгоритмом [22].

У роботі [23] запропоновано алгоритм *Generalized Tree Reduction*, який поєднує принципи дельта-дебагінгу з ієрархічними перетвореннями деревоподібних структур. Запропонований алгоритм автоматично скорочує складні вхідні дані до кількох відсотків від їх початкового розміру, істотно перевершуючи класичний дельта-дебагінг за ефективністю мінімізації.

Автори праці [24] запропонували ймовірнісний підхід до дельта-дебагінгу (*Probabilistic Delta Debugging*, ProbDD), який поєднує класичний алгоритм *ddmin* із побудовою ймовірнісної моделі на основі абстрактного синтаксичного дерева. Метод бере до уваги синтаксичні зв'язки між елементами й результати попередніх випробувань, що скорочує середній час на оброблення на 27% і зменшує розмір результатів у 3,4 раза порівняно з наявними методами.

Серед сучасних модифікацій дельта-дебагінгу варто згадати алгоритм дельта-дебагінгу *Weighted Delta Debugging* (WDD) [25], у якому під час мінімізації береться до уваги вага елементів вхідних даних. Така модифікація дає змогу розрізняти фрагменти за їх внеском у структуру тесту, що підвищує ефективність алгоритмів *ddmin* і *ProbDD*.

Зокрема в середньому час на мінімізацію скорочується до 51%, а розмір результатів на 13% менший, якщо порівнювати з основними методами.

Ідею *hoisting*-операцій для HDD розглянуто в статті [26], а глибший теоретичний аналіз стохастичного дельта-дебагінгу виконано в роботі [27].

Дельта-дебагінг адаптовано для регресійного тестування на основі атрибутів [28].

Окремо варто згадати нові модифікації, спрямовані на підвищення ефективності дельта-дебагінгу. Зокрема в роботі [29] запропоновано MIP-DD, що є першим відкритим інструментом на основі дельта-дебагінгу для задач змішаного цілочисельного програмування (*Mixed Integer Programming*, MIP).

У роботі [30] сучасна модифікація дельта-дебагінгу основана на ідеї ітерації до фіксованої точки (DDMIN*). Автори продемонстрували алгоритм, який додатково скорочує вхідні дані на 48% порівняно з класичним DDMIN, а також підвищує стабільність процесу мінімізації та зменшує кількість зайвих ітерацій.

Крім того, сучасні дослідження вказують на те, що навчання з підкріпленням може стати логічним продовженням еволюції жадібних і дельта-дебагінг алгоритмів [31-33]. Зокрема в студіях [34, 35] застосовано RL для формування ефективних ТС на основі агента з Q-навчанням.

2.4. Визначення проблеми

Отже, попри значний розвиток методів TSM, низка ключових питань залишається нерозв'язаною. Класичні жадібні та дельта-дебагінг алгоритми довели ефективність у зменшенні розміру ТН і часу

на виконання ТС, однак залишаються детермінованими й не використовують адаптивних стохастичних механізмів. Більшість наявних модифікацій оптимізують лише структурне покриття, не зважаючи на семантичну значущість дій ТС або взаємозалежність базових блоків виконання C++ бібліотек у складі ПЗ, що знижує діагностичну цінність отриманих результатів для C++ бібліотек.

Новітні модифікації, як-от імовірнісний і зважений дельта-дебагінг, підвищують ефективність мінімізації. Однак вони ґрунтуються на фіксованих евристиках і не пристосовані до тестування C++ бібліотек із високою структурною складністю.

Крім того, більшість методів передбачають послідовне оброблення елементів, тоді як реальні ТС містять розсіяні або контекстно залежні дії, важливі для збереження повноти покриття.

3. Мета й завдання дослідження

Метою роботи є підвищення ефективності тестування C++ бібліотек завдяки вдосконаленню класичного алгоритму жадібного формування (оптимізації) ТС і алгоритму дельта-дебагінг мінімізації ТС для C++ бібліотек.

Для досягнення поставленої мети в роботі сформульовано задачу формування (оптимізації) ТС для C++ бібліотек та визначено такі часткові наукові завдання:

1) удосконалення жадібного алгоритму оптимізації ТС та усунення його детермінованості із забезпеченням ефективного стиснення ТС для збереження гілкового покриття коду;

2) оптимізація алгоритму дельта-дебагінг мінімізації тестових сценаріїв для ТС із розрідженим розташуванням надлишкових дій;

3) оцінювання ефективності вдосконалених алгоритмів оптимізації ТС для C++ бібліотек порівняно з класичними алгоритмами.

З метою оцінювання ефективності необхідно попередньо визначити критерії ефективності алгоритмів формування (оптимізації) ТС для C++ бібліотек.

4. Матеріали й методи дослідження

Задачу формування ТС для C++ бібліотек без специфікацій API можна формалізувати як марковський процес прийняття рішень (*Markov Decision Process*, MDP) [36], де ТС будується крок за кроком, обираючи наступну функцію з допустимого простору дій (API) у такий спосіб:

$$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}(s' \mid s, a), \mathcal{R}(s, a), \gamma \rangle, \quad (1)$$

де $\mathcal{S}$ – множина станів, що відповідає стану s; $\mathcal{A}$ – множина допустимих дій у стані s; $\mathcal{P}(s' \mid s, a)$ – функція ймовірності переходу до нового стану після виконання дії a у стані s; $\mathcal{R}(s, a)$ – функція винагороди, яка надається за дію a в стані s; $\gamma \in (0,1)$ – коефіцієнт дисконтування.

Задача формування ТС описується таким чином. ТС довжини $n \in \mathbb{N}$, який формується, може бути впорядкованою послідовністю дій ТС:

$$S = (a_1, a_2, \ldots, a_n), \quad (2)$$

де $a_i \in \mathcal{A}$ – виклики API-бібліотеки з множини $\mathcal{A}$.

Щоб керувати формуванням ТС, додатково використовується покриття базових блоків (*basic-block coverage*) як евристика. Для кожної дії a, яка виконується у префіксі сценарію $S_{1:t}$, де $t \in \mathbb{N}$ – це поточний крок формування ТС, визначається множина блоків $BB(S_{1:t})$. За умови, якщо остання дія відкрила блок, який ніколи не виконувався, то дія відкриває унікальний базовий блок виконання програми. Індикатор унікальності можна записати таким чином:

$$u_t(a) = \begin{cases} 1, & якщо\, BB(S_{1:t}) / BB(S_{1:t-1}) \neq \varnothing \\ 0, & якщо\, BB(S_{1:t}) / BB(S_{1:t-1}) = \varnothing \end{cases}. \quad (3)$$

Миттєва корисність дії a на кроці t визначається за зростанням покриття гілок:

$$g_t(a) = Cov(S_{1:t} \oplus a) - Cov(S_{1:t}). \quad (4)$$

Тобто миттєва корисність дорівнює кількості нових гілок, які відкриються завдяки додаванню дії a на поточному кроці.

Задача мінімізації довжини ТС за умови збільшення (збереження) гілкового покриття. Необхідно знайти найкоротший тестовий сценарій S' на основі оригінального ТС S_{orig}, який задовольняє задану величину покриття коду:

$$\begin{cases} \min\limits_{S' \in \mathcal{A}} Len(S') \\ Cov(S') \geq Cov(S_{orig}) \end{cases}. \quad (5)$$

Розв’язати задачу (5) можна за допомогою формування нового тестового сценарію або мінімізації оригінального ТС за умови збереження оригінального покриття коду.

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 1 (35) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 1 (35) *ISSN 2524-2296 (online)*

4.2. Удосконалення жадібного алгоритму формування ТС

Одним із найбільш поширених підходів щодо формування або мінімізації ТС є використання жадібних алгоритмів. Основна ідея полягає в тому, щоб на кожному кроці обирати дію, яка забезпечує найбільший безпосередній приріст покриття. У класичній постановці це покриття може обчислюватися за різними критеріями. Наприклад, за кількістю нових вимог, які задовольняються, або – нових елементів коду, що активуються.

На кожному кроці t жадібного алгоритму – обчислюється миттєва корисність кожної, ще не використаної дії за формулою (3). Далі обирається дія з максимальною миттєвою корисністю:

$$a' = arg \max_{a \in \mathcal{A}_t} g_t\left(a\right). \quad (6)$$

За умови, якщо $g_t\left(a'\right) = 0$, то алгоритм завершує роботу, оскільки жодна з дій не дає змоги розширити гілкове покриття. В іншому разі обрана дія додається до сценарію та процес продовжується до досягнення оригінального покриття або до зупинки за іншим критерієм (наприклад, обмеження на довжину ТС). Псевдокод класичного жадібного алгоритму подано на рис. 1.

Попри свою простоту й практичну ефективність, класичний жадібний алгоритм має низку обмежень, а саме:

– *локальні плато* – якщо жодна з доступних дій не дає приросту покриття безпосередньо на поточному кроці, алгоритм зупиняється. Водночас він може втратити можливість досягти вищого покриття, яке відкривається лише після виконання певної "проміжної" дії;

– *детермінізм і відсутність диверсифікації* – алгоритм завжди обирає одну найкращу дію за миттєвим приростом, що робить його схильним до потрапляння в пастку локальних максимумів і обмежує різноманітність побудованих ТС;

– *ігнорування додаткової інформації* – класичний жадібний алгоритм не використовує жодної допоміжної інформації про потенційно корисні дії. Наприклад, він не бере до уваги, що деякі дії можуть відкривати нові базові блоки виконання програми, які згодом призведуть до зростання покриття гілок.

Класичний жадібний алгоритм обмежений тим, що на кожному кроці він відбирає лише дію з максимальним миттєвим приростом покриття гілок. Як наслідок, якщо всі доступні дії мають нульовий приріст, то алгоритм зупиняється, навіть якщо деякі з них могли б відкрити нові можливості для майбутнього покриття. Таке обмеження призводить до стагнації на локальних плато.

Алгоритм 1.
Класичний жадібний алгоритм формування ТС.

***Вхідні дані*:**
S_{orig} – оригінальний ТС.
Вихідні дані:
S_{out} – сформований ТС.
Тіло алгоритму:
1. $S_{out} \leftarrow \varnothing$
2. $\mathcal{A} \leftarrow Extract\left(S_{orig}\right)$
3. $Cov_{cur} \leftarrow \varnothing$
4. $Cov_{orig} \leftarrow Cov\left(S_{orig}\right)$
5. ***repeat***
6. $Gain_{best} \leftarrow 0$
7. $a' \leftarrow None$
8. ***foreach*** $a \in \mathcal{A} \backslash S_{out}$ ***do***
9. ***local*** $g\left(a\right) \leftarrow Cov\left(S_{out} \oplus a\right) - Cov_{cur}$
10. ***if*** $g\left(a\right) > Gain_{best}$ ***then***
11. $Gain_{best} \leftarrow g\left(a\right)$
12. $a' \leftarrow a$
13. ***end if***
14. ***if*** $a' = None$ ***or*** $Gain_{best} = 0$ ***then***
15. ***break***
16. ***end if***
17. ***end foreach***
18. $S_{out} \leftarrow S_{out} \oplus a$
19. $Cov_{cur} \leftarrow Cov\left(S_{out}\right)$
20. ***until*** $Cov_{cur} < Cov_{orig}$
21. ***return*** S_{out}

Рис 1. Псевдокод жадібного алгоритму формування ТС

Для подолання цього обмеження запропоновано вдосконалений жадібний алгоритм, що комбінує такі два параметри:

1) миттєвий приріст гілкового покриття $g_t\left(a\right)$;

2) майбутня корисність дії $f_t\left(a\right)$, яка оцінюється на основі появи нових унікальних базових блоків після виконання цієї дії.

Сутність удосконалення полягає в тому, що, якщо дія не збільшує покриття гілок у певний час, то вона може стати "ключем", після якого інші дії починають давати приріст. На кожному кроці вдосконалений алгоритм оцінює зважений бал дії:

$$F_t\left(a\right) = g_t\left(a\right) + \lambda \cdot f_t\left(a\right), \quad (7)$$

ISSN 2522-9818 (print) Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 1 (35)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2026. No. 1 (35)

де $\lambda \in [0;1]$ – вага, що визначає пам'ять про майбутню корисність дій ТС.

За умови, якщо виконання $S_{1:t} \oplus a$ породжує нові базові блоки, що не з'являлись раніше, то отримується індикатор унікальності дії (3) – $u_t(a) \in \{0,1\}$. Припускається, що після дії a з'явилась деяка множина дій:

$$\mathcal{N}(a) = \{b \in \mathcal{A} / u_t(b) = 1\}, \quad (8)$$

де $u_t(b) = 1$ означає, що дія a є "ключем".

Для таких дій зберігається майбутня корисність дії $f_t(a)$, яка оновлюється за таким правилом:

$$f_t(a) \leftarrow \max(f_t(a), \max_{b \in \mathcal{N}(a)} F_t(b)). \quad (9)$$

Отже, обрано оновлення майбутньої корисності за принципом максимального значення, оскільки для задачі формування ТС важливим є факт, що дія хоча б один раз відкрила нові унікальні базові блоки виконання програми. Використання максимального значення гарантує збереження найсильнішого спостереженого впливу дії, робить алгоритм більш стійким і спрощує його збіжність без потреби в додаткових параметрах. На кожній ітерації алгоритму формується множина кандидатів $\mathcal{A}_{cand} \in \mathcal{A}$, яка містить усі дії a, для яких виконується $u_t(a) = 1$ або $F_t(a) > 0$. З метою диверсифікації алгоритму можна застосовувати формулу *softmax* для вибору такої дії:

$$P_t(a) = \frac{e^{F_t(a)/\tau}}{\sum_{a^* \in \mathcal{A}_{cand}} e^{F_t(a^*)/\tau}}, \quad (10)$$

де τ – коефіцієнт, який керує балансом між жадібним вибором і дослідженням майбутньої користі дій ТС.

Псевдокод удосконаленого жадібного алгоритму подано на рис. 2.

Удосконалений жадібний алгоритм бере до уваги майбутню користь дій і здатний перевищити класичний жадібний алгоритм у задачі формування ТС. Завдяки збереженню інформації про дії, що відкривають доступ до нових унікальних станів, алгоритм має більший потенціал покриття гілок і очікувано формує коротші й ефективніші сценарії, ніж класичний алгоритм, що орієнтується лише на миттєвий приріст покриття.

Алгоритм 2.
Жадібний алгоритм формування ТС з огляду на майбутню користь дій ТС

***Вхідні дані*:**
– S_{orig} – оригінальний ТС;
– λ – вага майбутньої користі;
– τ – коефіцієнт *softmax* вибору дії;
– υ – кількість кращих кандидатів для вибору.

Вихідні дані:
– S_{out} – сформований ТС.

Тіло алгоритму:
1. $S_{out} \leftarrow \varnothing$, $\mathcal{A} \leftarrow Extract(S_{orig})$
2. $Cov_{orig} \leftarrow Cov(S_{orig})$
2. ***foreach*** $a \in \mathcal{A}$ ***do*** $f_t(a) \leftarrow 0$ ***end***
3. **repeat**
4. $\mathcal{A}_{cand} \leftarrow \varnothing$
5. ***foreach*** $a \in \mathcal{A} \setminus S_{out}$ ***do***
6. $g_t(a) \leftarrow Cov(S_{out} \oplus a) - Cov(S_{out})$
7. ***if*** $g_t(a) > 0$ **or** $f_t(a) > 0$ ***then***
8. $F_t(a) = g_t(a) + \lambda \cdot f_t(a)$
9. $\mathcal{A}_{cand} \leftarrow \mathcal{A}_{cand} \cup \{a\}$
10. ***end if***
11. ***end foreach***
12. ***if*** $\mathcal{A}_{cand} = \varnothing$ ***then break***
13. ***foreach*** *best* υ *candidates* $a \in \mathcal{A}_{cand}$ ***do***
14. $P_t(a) \leftarrow Equation_10(a, \tau)$
15. ***end foreach***
16. $a' \sim P_t(a)$
17. $S_{out} \leftarrow S_{out} \oplus a'$
18. ***foreach*** $b \in \mathcal{A} \setminus S_{out}$ ***do***
19. $F_t(b) \leftarrow g_t(b) + \lambda \cdot f_t(b)$
20. ***if*** $u_t(b) = 1$ ***and*** $F_t(b) > 0$ ***then***
21. $f_t(a) \leftarrow max(f_t(a), F_t(b))$
22. ***end if***
23. ***end foreach***
24. ***until*** $Cov(S_{out}) < Cov_{orig}$
25. ***return*** S_{out}

Рис. 2. Псевдокод жадібного алгоритму формування ТС з огляду на майбутню корисність дій

4.3. Удосконалення алгоритму дельта-дебагінг мінімізації тестових сценаріїв

Дельта-дебагінг є одним з найпоширеніших алгоритмів мінімізації ТС. Його сутність полягає в поступовому вилученні частин ТС і перевірці щодо збереження бажаної властивості, відтворення

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 1 (35) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 1 (35) *ISSN 2524-2296 (online)*

помилки або покриття коду. За умови, якщо скорочений ТС залишається валідним, то вилучені кроки вважаються надлишковими й відкидаються.

Псевдокод класичного алгоритму запропоновано на рис. 3, а додаткова процедура до алгоритму – на рис. 4.

Алгоритм 3.
Класичний алгоритм дельта-дебагінг мінімізації ТС

***Вхідні дані*:**
- S_{orig} – оригінальний ТС.

Вихідні дані:
- S_{out} – мінімізований ТС.

Тіло алгоритму:

1. $Cov_{orig} \leftarrow Cov\left(S_{orig}\right)$
2. $S_{out} \leftarrow S_{orig}$
3. $g \leftarrow 2$
4. ***while*** $Len\left(S_{out}\right) \geq 2$ ***do***
5. $\left\{S_1,\ldots,S_n\right\} \leftarrow PartitionContinous\left(S_{out},\ g\right)$
6. ***local*** $success \leftarrow false$
7. ***for*** $i \leftarrow 1$ ***to*** g ***do***
8. $S_{trial} \leftarrow S_{out} \setminus S_i$
9. ***if*** $Cov\left(S_{trial}\right) \geq Cov_{orig}$ ***then***
10. $S_{out} \leftarrow S_{trial}$
11. $g \leftarrow max\left(g-1, 2\right)$
12. $success \leftarrow true$
13. ***break***
14. ***end if***
15. ***end for***
16. ***if*** $success$ ***then continue***
17. ***for*** $i \leftarrow 1$ ***to*** g ***do***
18. $S_{trial} \leftarrow S_i$
19. ***if*** $Cov\left(S_{trial}\right) \geq Cov_{orig}$ ***then***
20. $S_{out} \leftarrow S_{trial}$
21. $g \leftarrow max\left(g-1, 2\right)$
22. $success \leftarrow true$
23. ***break***
24. ***end if***
25. ***end for***
26. ***if*** $success$ ***then continue***
27. ***if*** $g \geq Len\left(S_{out}\right)$ ***then***
28. ***break***
29. ***else***
30. $g \leftarrow min\left(2 \cdot g, Len\left(S_{out}\right)\right)$
31. ***end if***
32. ***end while***
33. ***return*** S_{out}

Рис. 3. Псевдокод класичного алгоритму дельта-дебагінг мінімізації ТС

Процедура 1.
Процедура розбиття ТС на послідовні блоки дій $PartitionContinous$

***Вхідні дані*:**
- S – послідовність дій;
- n – бажана кількість блоків.

Вихідні дані:
- $\left\{S_1,\ldots,S_n\right\}$ – суміжні блоки без перетинів.

Тіло процедури:

1. $Len_s \leftarrow Len\left(S\right)$
2. $n \leftarrow min\left(n,\ Len_s\right)$
3. $Len_{base} \leftarrow floor\left(Len_s\ /\ n\right)$
4. $Len_{rem} \leftarrow mod\left(Len_s\ /\ n\right)$
5. $base \leftarrow 1$
6. ***for*** $i \leftarrow 1$ ***to*** n ***do***
7. ***local*** $size \leftarrow base + \left(i \leq Len_{rem}\ ?1 : 0\right)$
8. $S_i \leftarrow Cut\left(S, base, -1\right]$
9. $base \leftarrow base + size$
10. ***end for***
11. ***return*** $\left\{S_1,\ldots,S_n\right\}$

Рис. 4. Псевдокод процедури розбиття ТС на послідовні блоки дій ТС

Алгоритм дельта-дебагінг працює за принципом поділу на підмножини. Тоді нехай вхідний ТС містить n дій. Алгоритм починає з розбиття множини дій на два блоки й перевіряє, чи зберігається покриття після вилучення одного з них.

За умови, якщо так, то відкидає блок. Якщо ні, тоді ділить на дрібніші частини (наприклад, чотири блоки) та повторює процес.

Отже, алгоритм гарантує поступову мінімізацію ТС до підмножини, яка забезпечує необхідний результат.

Для мінімізації ТС запропоновано вдосконалений варіант алгоритму дельта-дебагінгу, який поєднує класичне вилучення елементів із стратегією множинного вилучення дій та врахуванням унікальності базових блоків.

Отже, для кожної дії a_i визначається множина унікальних базових блоків програми:

$$U\left(a_i\right) = \left\{q \in B\left(a_i\right)/q \notin \bigcup_{j \neq i} B\left(a_j\right)\right\}, \qquad (11)$$

де $B\left(a_i\right)$ – множина базових блоків програми, покритих дією a_i.

За умови, якщо $U\left(a_i\right) \neq \varnothing$, то дія позначається як "захищена" й не потрапляє до кандидатів на

ISSN 2522-9818 (print) Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 1 (35)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2026. No. 1 (35)

вилучення поточної ітерації алгоритму. На кожній ітерації алгоритм формує підмножини дій без захищених елементів. Підмножини, вилучення яких не зменшує покриття нижче, ніж базовий рівень, вилучаються остаточно.

Псевдокод алгоритму запропоновано на рис. 5.

Процедуру формування множини захищених дій подано на рис. 6.

Алгоритм 4.
Алгоритм дельта-дебагінг мінімізації тестових сценаріїв з огляду на унікальність покриття базових блоків програми.

***Вхідні дані*:**
- S_{orig} – оригінальний ТС;
- $g_0 \in (0,1)$ – коефіцієнт зернистості;
- g_{min} – мінімальна зернистість.

Вихідні дані:
- S_{out} – мінімізований ТС.

Тіло алгоритму:

1. $Cov_{orig} \leftarrow Cov(S_{orig})$
2. $S_{out} \leftarrow S_{orig}$
3. $g \leftarrow g_0$
4. $st \leftarrow max(Len(S_{out}) \cdot g,\ g_{min})$
5. ***while*** $st \geq 1$ ***do***
6. $P \leftarrow ComputeProtected(S_{out})$
7. ***local*** $i \leftarrow 1$
8. ***while*** $i \leq Len(S_{out})$ ***do***
9. ***local*** $S_{batch} \leftarrow \varnothing$
10. $j \leftarrow i$
11. ***while*** $Len(B) < step$ ***and*** $j \leq Len(S_{out})$ ***do***
12. ***if*** $j \notin P$ ***then***
13. $S_{batch} \leftarrow B \cup \{a_j\}$
14. ***end if***
15. $j \leftarrow j+1$
16. ***end while***
17. $S_{trial} \leftarrow S_{out} \setminus S_{batch}$
18. ***if*** $Cov(S_{trial}) \geq Cov_{orig}$ ***then***
19. $S_{out} \leftarrow S_{trial}$
20. $P \leftarrow ComputeProtected(S_{out})$
21. ***else***
22. $i \leftarrow j$
23. ***end if***
24. ***end while***
25. $st \leftarrow max(st/2,\ g_{min})$
26. ***end while***
27. ***return*** S_{out}

Рис. 5. Псевдокод алгоритму дельта-дебагінг мінімізації ТС з огляду на унікальність покриття базових блоків програми

Алгоритм 4 керується $g \in \mathbb{R}$ – коефіцієнтом зернистості, що визначає кількість підмножин, на які розбивається вхідна послідовність дій.

На початкових ітераціях алгоритму це дає змогу виконувати більш грубі спроби вилучення, а в разі невдачі коефіцієнт зернистості поступово зростає, забезпечуючи більш дрібнозернисте розбиття. Такий підхід дає змогу балансувати між швидкістю мінімізації та її ефективністю.

Процедура 2.
Процедура формування множини дій за унікальним внеском у покриття базових блоків програми $ComputeProtected$

***Вхідні дані*:**
- S – послідовність дій.

Вихідні дані:
- P – множина "захищених" дій.

Тіло процедури:

1. $P \leftarrow \varnothing$
2. ***for*** $j=1$ ***to*** $Len(S)$ ***do***
3. ***if*** $U(a_j) \neq \varnothing$ ***then***
4. $P \leftarrow P \cup \{a_j\}$
5. ***end if***
6. ***end for***
7. ***return*** P

Рис. 6. Псевдокод процедури формування множини "захищених" дій

Запропонований алгоритм має потенційні переваги над класичним дельта-дебагінгом у задачі формування ТС для C++ бібліотек завдяки адаптивному керуванню значенням коефіцієнта g врахуванню унікальності дій ТС за допомогою обчислення покриття базових блоків виконання програми. Використання змінної зернистості дає змогу спочатку швидко відсіювати великі непотрібні фрагменти ТС, а потім поступово переходити до більш точного локального аналізу, що забезпечує баланс між швидкістю мінімізації та якістю кінцевого результату.

Додаткове врахування дій із унікальним внеском у покриття базових блоків запобігає втраті критичних елементів ТС. Така комбінація дає змогу досягати високого ступеня стиснення ТС без зниження рівня покриття, особливо в разі довгих і надлишкових ТС.

4.4. Критерії ефективності алгоритмів оптимізації ТС

ТН C++ бібліотеки може бути поданий як впорядкована послідовність тестових сценаріїв:

$$TS = (S_1, S_2, \ldots, S_m), \quad (12)$$

де m – кількість ТС у ТН.

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 1 (35) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 1 (35) *ISSN 2524-2296 (online)*

Для оцінювання ефективності алгоритмів оптимізації ТС використано три основні метрики [4].

1. Коефіцієнт збереження покриття (*Retention of Coverage*, RC) – оцінює приріст гілкового покриття ТН, що відтворює втрату (збереження) ефективності тестування:

$$RC = Cov(TS_{out}) / Cov(TS_{orig}), \qquad (13)$$

де $Cov(TS_{out})$ – покриття ТН після оптимізації; $Cov(TS_{orig})$ – покриття оригінального ТН.

2. Коефіцієнт стиснення (*Compression Ratio*, CR) – оцінює зменшення кількості інструкцій ТН:

$$CR = \left(Len\left(TS_{orig}\right) - Len\left(TS_{out}\right)\right) / Len\left(TS_{orig}\right), \qquad (14)$$

де $Len\left(TS_{orig}\right)$ – кількість інструкцій оригінального ТН; $Len\left(TS_{out}\right)$ – після оптимізації.

3. Коефіцієнт збереження часу (*Execution Cost Reduction*, ECR) – оцінює зменшення часу на виконання ТН:

$$ECR = \left(T\left(TS_{orig}\right) - T\left(TS_{out}\right)\right) / T\left(TS_{orig}\right), \qquad (15)$$

де $T\left(TS_{orig}\right)$ – час на виконання оригінального тестового набору; $T\left(TS_{out}\right)$ – час на виконання ТН після оптимізації.

5. Результати дослідження та їх обговорення

5.1. Налаштування середовища оцінювання

Для оцінювання ефективності вдосконалених алгоритмів обрано дві C++ бібліотеки з відкритим кодом, які різняться за масштабом і структурною складністю.

Основні властивості подано в табл. 1.

Таблиця 1. *Властивості C++ бібліотек, обраних для дослідження*

Бібліотека	API	LoC	m	IC	BC %	Avg. CCN
Bitmap PlusPlus	33	760	7	71	27,0	2,5
Hjson	122	3936	57	1259	36,1	5,4

У табл. 1 використані такі позначки:

– API – кількість публічних функцій;
– LoC – кількість рядків у вихідному коді;
– m – кількість ТС у ТН;
– IC – загальна кількість інструкцій у ТН;
– BC – гілкове покриття ТН;
– Avg. CCN – середня цикломатична складність функцій бібліотеки.

Оцінку виконано на робочій станції з процесором Intel Core i7 (6 ядер, 2.6 ГГц) та 16 Гб оперативної пам’яті. Компіляцію бібліотек і ТС здійснено за допомогою AppleClang 15.0.0 (LLVM 15) з увімкненими параметрами профілювання покриття

-fprofile-instr-generate fcoverage-mapping.

Для збору та аналізу покриття використано утиліту

llvm-profdata та *llvm-cov*.

Крім того, застосовано механізм збору інформації для ідентифікації базових блоків, що активовані під час виконання кожної дії ТС:

__llvm_profile_instrument_target.

Зібрані дані перетворено у двійкові вектори покриття, які асоціюються з відповідними діями ТС.

Для аналізу кількості рядків коду й цикломатичної складності використано утиліту *lizard.*

5.2. Конфігурації досліджуваних алгоритмів

Для порівняльного аналізу використано дві групи конфігурацій алгоритмів і два базові алгоритми:

– конфігурації вдосконаленого жадібного алгоритму формування ТС (GRR-TD1, GRR-TD2, GRR-TD3), що реалізують алгоритм 2;

– конфігурації вдосконаленого алгоритму дельта-дебагінгу оптимізації ТС (DSL-FM1, DSL-FM2, DSL-FM3), що реалізують алгоритм 4;

– конфігурація класичного жадібного алгоритму формування ТС (GR), що реалізує алгоритм 1;

– конфігурація класичного алгоритму дельта-дебагінг мінімізації ТС (DSL), що реалізує алгоритм 3.

Для кожної конфігурації алгоритму проведено 100 незалежних математичних моделювань послідовно для кожного тестового сценарію з ТН цільових C++ бібліотек.

Конфігурації вдосконаленого жадібного алгоритму формування ТС подано в табл. 2.

У табл. 2 значення параметра τ у механізмі *softmax* вибору дій зафіксовано на рівні 1,5, що обґрунтовано компромісом між жадібністю та можливістю дослідити вплив майбутньої корисності.

Конфігурації вдосконаленого алгоритму дельта-дебагінг мінімізації ТС подано в табл. 3.

ISSN 2522-9818 (print) *Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 1 (35)*
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2026. No. 1 (35)*

Таблиця 2. *Конфігурації вдосконаленого жадібного алгоритму оптимізації ТС*

Конфігурація	Вага майбут. корисності, λ	Розмір множ. канд. υ	Параметр *Softmax*, τ
GRR-TD1	0,35	3	1,5
GRR-TD2	0,65	4	1,5
GRR-TD3	0,9	5	1,5

Таблиця 3. *Конфігурації вдосконаленого алгоритму дельта-дебагінг мінімізації ТС*

Конфігурація	Коефіцієнт зернистості, g_0	Мінімальна зернистість, g_{min}
DSL-FM1	0,20	5
DSL-FM2	0,25	3
DSL-FM3	0,30	1

5.3. Аналіз збереження покриття ТН

На рис. 7 зображено розподіл показників гілкового покриття коду залежно від конфігурації алгоритму оптимізації ТС для С++ бібліотек *BitmapPlusPlus* та *Hjson.* Пунктирна лінія позначає рівень покриття оригінального ТН.

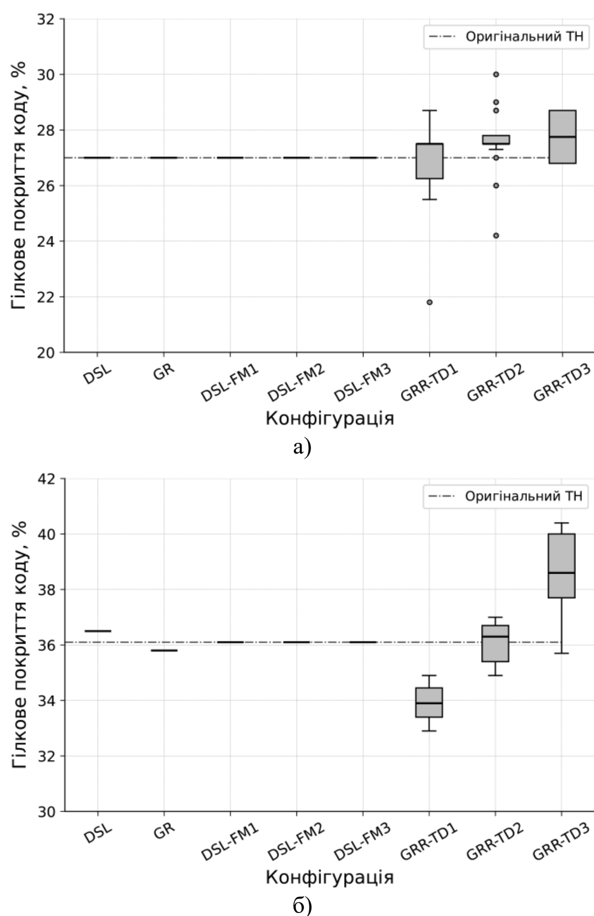


Рис. 7. Розподіл показників гілкового покриття коду залежно від конфігурації алгоритму оптимізації ТС для бібліотек *BitmapPlusPlus* (а) та *Hjson* (б)

Розподіл покриття для бібліотеки *BitmapPlusPlus* (рис. 7, *а*) вказує на те, що класичні алгоритми DSL і GR, а також усі конфігурації DSL-FM стабільно зберігають початковий рівень гілкового покриття. Натомість удосконалені конфігурації GRR-TD демонструють більшу варіативність результатів і підвищене медіанне покриття. Отже, врахування майбутньої корисності дій дає змогу алгоритму компенсувати втрату покриття під час скорочення ТС, утримуючи або навіть підвищуючи досяжність гілок.

На рівні розподілу покриття можна спостерігати стабільність DSL і адаптивну поведінку вдосконалених GRR-TD конфігурацій, що підвищують гілкове покриття коду зі зростанням складності середовища тестування.

5.4. Аналіз стиснення ТН

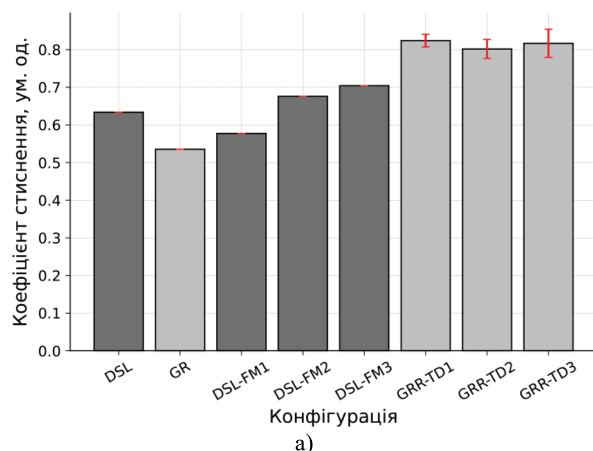
На рис. 8 наведено залежність коефіцієнта стиснення ТС від конфігурації алгоритмів для бібліотек *BitmapPlusPlus* і *Hjson*. Червоні відрізки на стовпчиках позначають стандартну похибку.

Для бібліотеки *BitmapPlusPlus* (рис. 8, *а)* спостерігається стабільне зростання коефіцієнта стиснення від базових конфігурацій (DSL, GR) до запропонованих (DSL-FM, GRR-TD). Значення підвищуються приблизно від 0,63 до 0,86, що свідчить про збільшення стиснення ТС. Особливо ефективними виявилися конфігурації жадібного формування з огляду на майбутню корисність дій (GRR-TD), які забезпечують не лише найбільший коефіцієнт стиснення, а й високий ступінь збільшення гілкового покриття коду.

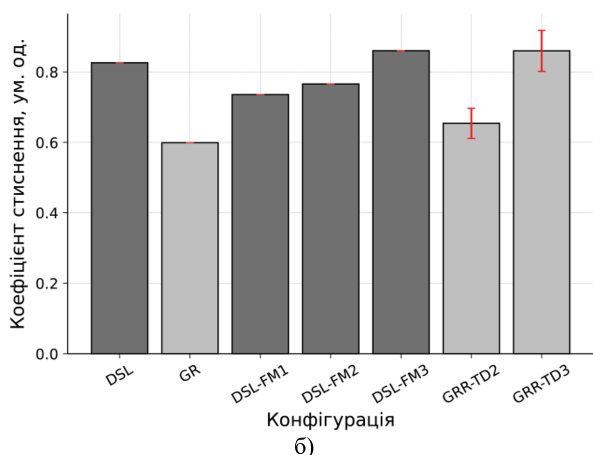
Для бібліотеки *Hjson* (рис. 8, *б*) результати виявляються більш варіативними. Базовий DSL демонструє високий коефіцієнт стиснення (~0,83), тоді як базовий GR значно нижчий (~0,6), що

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 1 (35) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 1 (35) *ISSN 2524-2296 (online)*

пояснюється більш складною структурою викликів API в обраній бібліотеці та основними недоліками жадібного алгоритму. Удосконалені конфігурації DSL-FM поступово підвищують коефіцієнт до ~0,86, що вказує на ефективність фільтрації неінформативних дій.



а)



б)

Рис. 8. Коефіцієнт стиснення залежно від конфігурації оптимізації ТС для бібліотек *BitmapPlusPlus* (а) та *Hjson* (б)

Натомість для конфігурацій GRR-TD отримано такі результати:

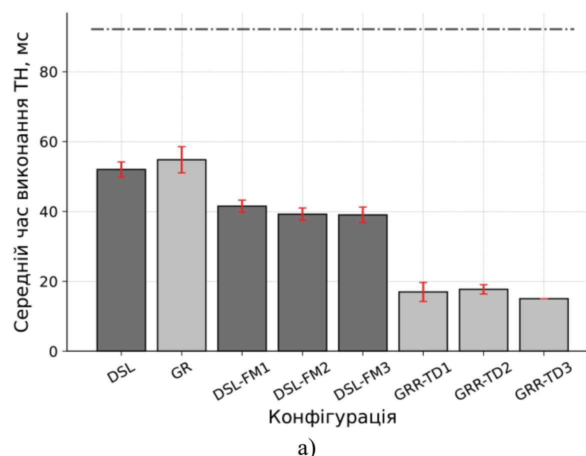
– GRR-TD1 – не подана на графіку через нездатність забезпечити адекватний рівень гілкового покриття;

– GRR-TD2 – досягає кращого стиснення, ніж класичний GR, але визначається збільшенням похибки стиснення;

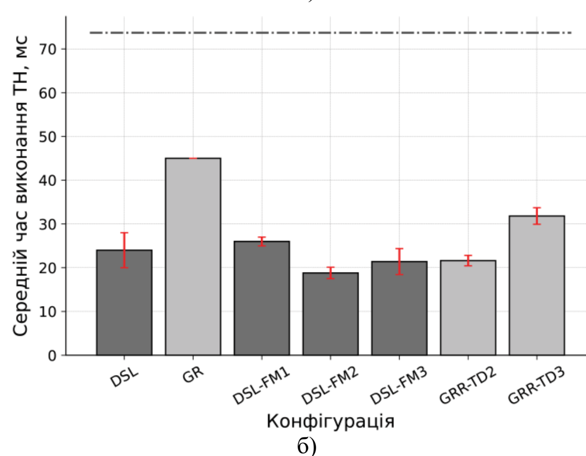
– GRR-TD3 – має найвище значення коефіцієнта стиснення (0,86), проте визначається великою похибкою.

5.5. Аналіз зменшення часу на виконання ТН

Результати аналізу часу на виконання ТН подано на рис. 9. Вони показують середній час на виконання ТН для різних конфігурацій під час тестування цільових бібліотек. Червоні відрізки на стовпчиках позначають стандартну похибку. Пунктирна лінія відповідає часу на виконання оригінального ТН, що використаний як орієнтир.



а)



б)

Рис. 9. Середній час на виконання ТН залежно від конфігурації оптимізації ТС для бібліотек: *BitmapPlusPlus* (а) та *Hjson* (б)

На рис. 9, *а* для бібліотеки *BitmapPlusPlus* спостерігається зменшення часу на виконання від базових конфігурацій (DSL, GR) до вдосконалених конфігурацій (DSL-FM, GRR-TD). Середній час на виконання знижується приблизно з 92 мс до 20 мс, тобто більше ніж утричі.

Аналогічна тенденція спостерігається й для бібліотеки *Hjson* (рис. 9, *б*), де час на виконання скорочується понад удвічі.

Конфігурація GR має найбільший середній час (~45 мс), тоді як удосконалені конфігурації DSL-FM і GRR-TD2 забезпечують стабільно нижчі показники – у межах від 19 мс до 25 мс.

Конфігурація GRR-TD3 має дещо більший час (~32 мс), що може бути наслідком складнішої логіки

перевірок під час виконання. Однак вона залишається загалом ефективнішою за базові конфігурації.

5.6. Загальна оцінка ефективності алгоритмів

У табл. 4 й 5 наведено зведені результати оцінювання ефективності конфігурацій досліджуваних алгоритмів для C++ бібліотек *BitmapPlusPlus* і *Hjson* відповідно.

Таблиця 4. *Оцінка ефективності конфігурацій досліджуваних алгоритмів для бібліотеки BitmapPlusPlus*

Конфіг.	Коеф. збер. покриття, RC	Коеф. стиснення, CR	Коеф. збер. часу, ECR
DSL	1	0,63	0,34±0,013
GR	1	0,54	0,28±0,048
DSL-FM1	1	0,58	0,44±0,02
DSL-FM2	1	0,68	0,55±0,021
DSL-FM3	1	0,70	0,51±0,024
GRR-TD1	0,92±0,026	0,82±0,016	0,84±0.04
GRR-TD2	1,023±0,03	0,80±0,025	0,82±0,02
GRR-TD3	1,0285±0,038	0,82±0,037	0,86±0,001

Таблиця 5. *Оцінка ефективності конфігурацій досліджуваних алгоритмів для бібліотеки Hjson*

Конфіг.	Коеф. збер. покриття, RC	Коеф. стиснення, CR	Коеф. збер. часу, ECR
DSL	1,011	0,83	0,61±0,07
GR	0,991	0,60	0,228±0,001
DSL-FM1	0,99	0,74	0,61±0,011
DSL-FM2	0,99	0,77	0,67±0,012
DSL-FM3	1	0,85	0,63±0,042
GRR-TD1	0,93±0,02	0,00	0,66±0,038
GRR-TD2	0,988±0,02	0,65±0,04	0,7±0,024
GRR-TD3	1,058±0,046	0,86±0,06	0,55±0,04

Для бібліотеки *BitmapPlusPlus* (табл. 4) базові конфігурації DSL і GR мають однакове збереження покриття (RC = 1), але відносно невелике стиснення (CR = 0,63 та 0,54) і низьке зменшення часу на виконання (ECR ≈ 0,34 та 0,28).

Удосконалені конфігурації DSL-FM забезпечують покращення CR до 0,7 та збільшення ECR до 0,55, що свідчить про ефективне вилучення надлишкових дій.

Найкращі результати демонструють жадібні конфігурації з огляду на майбутню корисність GRR-TD2 і GRR-TD3, які досягають максимальних значень:

$$RC \approx 1{,}0285,\ CR \approx 0{,}82 \text{ та } ECR \approx 0{,}86.$$

Отже, спостерігається динамічне скорочення ТС після досягнення оригінального гілкового покриття й забезпечення найбільшого зменшення часу на виконання ТН за умови повного збереження ефективності тестування.

Для бібліотеки *Hjson* (табл. 5) конфігурація DSL зберігає (збільшує) повне покриття (RC = 1,011) та забезпечує високе зменшення часу на виконання (ECR ≈ 0,61), що робить його основним орієнтиром.

Жадібна конфігурація GR має трохи менший коефіцієнт збереження покриття (RC = 0,991) і помітно нижчий коефіцієнт ECR ≈ 0,228, завдяки чому він поступається за ефективністю.

Конфігурації DSL-FM демонструють стабільні результати. У цьому разі значення коефіцієнтів ефективності лежать у таких інтервалах:

$$RC \in [0{,}99;1],\ CR \in [0{,}74;0{,}85] \text{ та } ECR \in [0{,}63;0{,}67].$$

Отримані дані підтверджують ефективність запропонованих конфігурацій порівняно з базовими. Для конфігурацій GRR-TD2 і GRR-TD3 значення RC перевищують 1 (≈1,058), що означає підвищення гілкового покриття. Конфігурація GRR-TD2 досягає найвищого показника ECR у середньому 0,7, а GRR-TD3 – найвищого коефіцієнта стиснення у середньому 0,86, але з великим розкидом значень.

6. Висновки

У статті запропоновано вдосконалення класичного алгоритму жадібного формування (оптимізації) ТС і алгоритму дельта-дебаггінг мінімізації ТС, а також оцінено їх ефективність.

Удосконалений жадібний алгоритм формування (оптимізації) ТС дає змогу підвищити коефіцієнт стиснення ТС з 0,63 до 0,86 і скоротити час на виконання ТН у ~2,5 раза, якщо порівнювати з класичним жадібним алгоритмом, за умови повного зростання (збереження) гілкового покриття з коефіцієнтом до 1,058.

Удосконалений алгоритм дельта-дебагінг, на відміну від класичного алгоритму, демонструє зменшення часу на виконання ТН у ~1,5 раза, з коефіцієнтом стиснення до 0,85, а також повним збереженням покриття.

Отже, до основних наукових досягнень дослідження належать такі:

– вдосконалено жадібний алгоритм, у якому дії ТС обираються з огляду на їх майбутню корисність, визначену за очікуваним приростом гілкового покриття коду. Такий підхід усуває детермінованість класичного жадібного алгоритму, підвищує

інформативність тестового сценарію та забезпечує збалансованість між збереженням гілкового покриття та стисненням ТС;

– вдосконалено алгоритм дельта-дебагінгу, який виконує групове вилучення неунікальних дій ТС за внеском у гілкове покриття, що дає змогу суттєво знизити довжину тестового сценарію без втрати властивостей ефективності тестування.

З практичного погляду, ефективна оптимізація ТС є важливою для тестових сценаріїв, спрямованих на виявлення вразливостей у смарт-контрактах на основі моделей і методів глибокого машинного навчання [37], а також для тестування коректності та стійкості криптографічних алгоритмів шифрування в прикладному ПЗ [38].

Отже, отримані результати можна використати для підвищення ефективності тестування C++ бібліотек із застосуванням CI/CD-інструментів автоматизованого тестування в промислових середовищах розробки ПЗ, зокрема для оптимізації ТН у процесах безперервної інтеграції та регресійного тестування C++ бібліотек у складі ПЗ з високими вимогами до якості й безпеки.

Подальші дослідження необхідно спрямувати на поєднання запропонованих алгоритмів з агентними методами на основі навчання з підкріпленням для створення адаптивних систем формування ТС для C++ бібліотек [31–33].

Конфлікт інтересів

Автори декларують, що не мають конфлікту інтересів, зокрема фінансового, особистого, авторського або будь-якого іншого характеру, який міг би вплинути на дослідження, а також на результати, опубліковані в цій статті.

Фінансування

Дослідження проводилося без фінансової підтримки.

Доступність даних

Дані будуть надані за обґрунтованим запитом.

Використання засобів штучного інтелекту

Автори підтверджують, що не застосовували технології штучного інтелекту для написання статті.

References

1. Hulevych, M. (2024), "CIDER: Assisted automation tool for C++ libraries testing", *Control, Navigation and Communication Systems*, Vol. 2, No. 76, pp. 74–77. DOI: https://doi.org/10.26906/sunz.2024.2.074
2. Hulevych, M., Kolomiitsev, O. (2025), "Automated test generation techniques for C++ software", *Control, Navigation and Communication Systems*, Vol. 2, No. 80, pp. 102–107. DOI: https://doi.org/10.26906/SUNZ.2025.2.102
3. Tallam, S., Gupta, N. (2005), "A concept analysis inspired greedy algorithm for test suite minimization", in *Proceedings of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE '05)*, Association for Computing Machinery, New York, NY, USA, pp. 35–42. DOI: https://doi.org/10.1145/1108792.1108802
4. Rehman Khan, S.U., Lee, S.P., Javaid, N., Abdul, W. (2018), "A Systematic Review on Test Suite Reduction: Approaches, Experiment's Quality Evaluation, and Guidelines", *IEEE Access*, Vol. 6, pp. 11816–11841. DOI: https://doi.org/10.1109/ACCESS.2018.2809600
5. Yoo, S., Harman, M. (2012), "Regression testing minimization, selection and prioritization: A survey", *Software Testing, Verification and Reliability*, Vol. 22, No. 2, pp. 67–120. DOI: https://doi.org/10.1002/stvr.430
6. Noemmer, R., Haas, R. (2020), "An Evaluation of Test Suite Minimization Techniques", in Winkler, D., Biffl, S., Mendez, D. and Bergsmann, J. (Eds.), *Software Quality: Quality Intelligence in Software and Systems Engineering (SWQD 2020)*, Springer, Vol. 371, pp. 51–66. DOI: https://doi.org/10.1007/978-3-030-35510-4_4
7. Li, Z., Harman, M., Hierons, R.M. (2007), "Search Algorithms for Regression Test Case Prioritization", *IEEE Transactions on Software Engineering*, Vol. 33, No. 4, pp. 225–237. DOI: https://doi.org/10.1109/TSE.2007.38
8. Yoo, S., Harman, M., Tonella, P., Susi, A. (2009), "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge", in *Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA 2009)*, Association for Computing Machinery, New York, NY, USA, pp. 201–212. DOI: https://doi.org/10.1145/1572272.1572296
9. Wang, S., Ali, S., Gotlieb, A., Liaaen, M. (2016), "A systematic test case selection methodology for product lines: Results and insights from an industrial case study", *Empirical Software Engineering*, Vol. 21, pp. 1586–1622. DOI: https://doi.org/10.1007/s10664-014-9345-5

10. Parsa, S., Khalilian, A. (2009), "A Bi-objective Model Inspired Greedy Algorithm for Test Suite Minimization", in Lee, Yh., Kim, Th., Fang, Wc. and Ślęzak, D. (Eds.), *Future Generation Information Technology (FGIT 2009)*, Springer, Vol. 5899, pp. 208–215. DOI: https://doi.org/10.1007/978-3-642-10509-8_24
11. Jehan, S., Wotawa, F. (2023), "An Empirical Study of Greedy Test Suite Minimization Techniques Using Mutation Coverage", *IEEE Access*, Vol. 11, pp. 65427–65442. DOI: https://doi.org/10.1109/ACCESS.2023.3289073
12. Putra, A.W., Legowo, N. (2025), "Greedy Algorithm Implementation for Test Case Prioritization in the Regression Testing Phase", *Journal of Computer Science*, Vol. 21, No. 2, pp. 290–303. DOI: https://doi.org/10.3844/jcssp.2025.290.303
13. Górski, T. (2025), "Pattern-Based Test Suite Reduction Method for Smart Contracts", *Applied Sciences*, Vol. 15, No. 2, Article 620. DOI: https://doi.org/10.3390/app15020620
14. Lin, C.-T., Tang, K.-W., Wang, J.-S., Kapfhammer, G.M. (2017), "Empirically evaluating Greedy-based test suite reduction methods at different levels of test suite complexity", *Science of Computer Programming*, Vol. 150, pp. 1–25. DOI: https://doi.org/10.1016/j.scico.2017.05.004
15. Habib, A.S., Khan, S.U.R., Felix, E.A. (2023), "A systematic review on search-based test suite reduction: State-of-the-art, taxonomy, and future directions", *IET Software*, Vol. 17, No. 2, pp. 93–136. DOI: https://doi.org/10.1049/sfw2.12104
16. Li, F., Zhou, J., Li, Y., Hao, D., Zhang, L. (2022), "AGA: An Accelerated Greedy Additional Algorithm for Test Case Prioritization", *IEEE Transactions on Software Engineering*, Vol. 48, No. 12, pp. 5102–5119. DOI: https://doi.org/10.1109/TSE.2021.3137929
17. Pan, R., Ghaleb, T.A., Briand, L.C. (2024), "LTM: Scalable and Black-Box Similarity-Based Test Suite Minimization Based on Language Models", *IEEE Transactions on Software Engineering*, Vol. 50, No. 11, pp. 3053–3070. DOI: https://doi.org/10.1109/TSE.2024.3469582
18. Zhang, Q., Fang, C., Sun, W., Yu, S., Xu, Y., Liu, Y. (2022), "Test case prioritization using partial attention", *Journal of Systems and Software*, Vol. 192, Article 111419. DOI: https://doi.org/10.1016/j.jss.2022.111419
19. Zeller, A., Hildebrandt, R. (2002), "Simplifying and isolating failure-inducing input", *IEEE Transactions on Software Engineering*, Vol. 28, No. 2, pp. 183–200. DOI: https://doi.org/10.1109/32.988498
20. Cleve, H., Zeller, A. (2005), "Locating causes of program failures", in *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, Association for Computing Machinery, New York, NY, USA, pp. 342–351. DOI: https://doi.org/10.1145/1062455.1062522
21. Harman, M., Hierons, R. (2001), "An overview of program slicing", *Software Focus*, Vol. 2, pp. 85–92. DOI: https://doi.org/10.1002/swf.41
22. Misherghi, G., Su, Z. (2006), "HDD: Hierarchical delta debugging", in *Proceedings of the 28th International Conference on Software Engineering (ICSE 2006)*, Association for Computing Machinery, New York, NY, USA, pp. 142–151. DOI: https://doi.org/10.1145/1134285.1134307
23. Herfert, S., Patra, J., Pradel, M. (2017), "Automatically reducing tree-structured test inputs", in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*, IEEE, Urbana, IL, USA, pp. 861–871. DOI: https://doi.org/10.1109/ASE.2017.8115697
24. Wang, G., Wu, Y., Zhu, Q., Xiong, Y., Zhang, X., Zhang, L. (2023), "A Probabilistic Delta Debugging Approach for Abstract Syntax Trees", in *Proceedings of the 34th IEEE International Symposium on Software Reliability Engineering (ISSRE 2023)*, IEEE, Florence, Italy, pp. 763–773. DOI: https://doi.org/10.1109/ISSRE59848.2023.00060
25. Zhou, X., Xu, Z., Zhang, M., Tian, Y., Sun, C. (2025), "WDD: Weighted Delta Debugging", in *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE 2025)*, IEEE, Ottawa, ON, Canada, pp. 1592–1603. DOI: https://doi.org/10.1109/ICSE55347.2025.00071
26. Vince, D., Hodován, R., Bársony, D., Kiss, Á. (2022), "The effect of hoisting on variants of Hierarchical Delta Debugging", *Journal of Software: Evolution and Process*, Vol. 34, No. 11, Article e2483. DOI: https://doi.org/10.1002/smr.2483
27. Zhang, M., Xu, Z., Tian, Y., Cheng, X., Sun, C. (2025), "Toward a Better Understanding of Probabilistic Delta Debugging", in *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE 2025)*, IEEE, Ottawa, ON, Canada, pp. 2024–2035. DOI: https://doi.org/10.1109/ICSE55347.2025.00117
28. Pontolillo, G.J., Mousavi, M.R. (2024), "Delta Debugging for Property-Based Regression Testing of Quantum Programs', in *Proceedings of the 5th ACM/IEEE International Workshop on Quantum Software Engineering (Q-SE 2024)*, Association for Computing Machinery, New York, NY, USA, pp. 1–8. DOI: https://doi.org/10.1145/3643667.3648219
29. Hoen, A., Kamp, D., Gleixner, A. (2025), "MIP-DD: Delta Debugging for Mixed-Integer Programming Solvers", *INFORMS Journal on Computing*. Advance online publication. DOI: https://doi.org/10.1287/ijoc.2024.0844
30. Vince, D., Kiss, Á. (2024), "Evaluation of the fixed-point iteration of minimizing delta debugging", *Journal of Software: Evolution and Process*, Vol. 36, No. 10, Article e2702. DOI: https://doi.org/10.1002/smr.2702

31. Kuchuk, N., Kashkevich, S., Radchenko, V., Andrusenko, Y., Kuchuk, H. (2024), "Applying edge computing in the execution IoT operative transactions", *Advanced Information Systems*, Vol. 8, No. 4, pp. 49–59. DOI: https://doi.org/10.20998/2522-9052.2024.4.07
32. Durmaz, E., Tümer, M.B. (2022), "Intelligent software debugging: A reinforcement learning approach for detecting the shortest crashing scenarios", *Expert Systems with Applications*, Vol. 198, Article 116722. DOI: https://doi.org/10.1016/j.eswa.2022.116722
33. Kuchuk, H., Kalinin, Y., Dotsenko, N., Chumachenko, I., Pakhomov, Y. (2024), "Decomposition of integrated high-density IoT data flow", *Advanced Information Systems*, Vol. 8, No. 3, pp. 77–84. DOI: https://doi.org/10.20998/2522-9052.2024.3.09
34. Semenov, S., Kolomiitsev, O., Hulevych, M., Mazurek, P., Chernyk, O. (2025), "An Intelligent Method for C++ Test Case Synthesis Based on a Q-Learning Agent", *Applied Sciences*, Vol. 15, No. 15, Article 8596. DOI: https://doi.org/10.3390/app15158596
35. Hulevych M. (2025), "Evaluation of the effectiveness of the test scenarios forming method for C++ libraries based on a Q-learning agent", *Management Information Systems and Devises*, No. 4 (187). pp. 20–46. DOI: https://doi.org/10.30837/0135-1710.2025.187.020
36. Puterman, M.L. (1994), *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York. DOI: https://doi.org/10.1002/9780470316887
37. Shmatko O., Kolomiitsev O., Rekova N., Kuchuk N., Matvieiev O. (2023), "Designing and Evaluating DL-Model for Vulnerability Detection in Smart Contracts", *Advanced Information Systems*, Vol. 7, No. 4, pp. 41–51. DOI: https://doi.org/10.20998/2522-9052.2023.4.05
38. Fedorchenko V., Yeroshenko O., Shmatko O., Kolomiitsev O., Omarov M. (2024), "Password Hashing Methods and Algorithms on the .NET Platform", *Advanced Information Systems*, Vol. 8, No. 4, pp. 82–92. DOI: https://doi.org/10.20998/2522-9052.2024.4.11

Received (Надійшла) 20.12.2025
Accepted for publication (Прийнята до друку) 10.02.2026
Publication date (Дата публікації) 30.03.2026

Відомості про авторів / About the Authors

Коломійцев Олексій Володимирович – заслужений винахідник України, доктор технічних наук, професор, Національний технічний університет "Харківський політехнічний інститут", професор кафедри комп'ютерної інженерії та програмування, Харків, Україна;
Oleksii Kolomiitsev – Honored Inventor of Ukraine, Doctor of Technical Sciences, Professor, National Technical University "Kharkiv Polytechnic Institute", Professor at the Department of Computer Engineering and Programming, Kharkiv, Ukraine;
e-mail: alexus_k@ukr.net
ORCID ID: http://orcid.org/0000-0001-8228-8404
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=57211278112

Гулевич Михайло Володимирович – аспірант, Національний технічний університет "Харківський політехнічний інститут", кафедра комп'ютерної інженерії та програмування, Харків, Україна;
Mykhailo Hulevych – PhD Student, National Technical University "Kharkiv Polytechnic Institute", Department of Computer Engineering and Programming, Kharkiv, Ukraine;
e-mail: gulevich30misha@gmail.com
ORCID ID: http://orcid.org/0009-0003-8622-3271
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=60039563700

Дмітрієв Олег Миколайович – доктор технічних наук, професор, Державний науково-дослідного інститут випробувань і сертифікації озброєння та військової техніки, провідний науковий співробітник, Черкаси, Україна;
Oleh Dmitriiev – Doctor of Technical Sciences, Professor, Leading Researcher, State Scientific Research Institute of Armament and Military Equipment Testing and Certification, Leading Researcher, Cherkasy, Ukraine;
e-mail: dmitriievoleh@gmail.com
ORCID ID: http://orcid.org/0000-0003-1079-9744
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=57208337326

ISSN 2522-9818 (print) *Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 1 (35)*
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2026. No. 1 (35)*

Левченко Андрій Олександрович – кандидат технічних наук, доцент, Військова академія (м. Одеса), професор кафедри застосування підрозділів військової розвідки та Сил спеціальних операцій, Одеса, Україна;

Andrii Levchenko – Candidate of Technical Sciences, Associate Professor, Odessa Military Academy, Professor at the Department of Training of Intelligence Units and Special Operations Forces, Odesa, Ukraine;

e-mail: katyaandreylev@gmail.com

ORCID ID: https://orcid.org/0000-0001-5550-0027

Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=57220805603

Балабуха Олексій Сергійович – кандидат технічних наук, Харківський національний університет Повітряних Сил ім. І. Кожедуба, докторант, Харків, Україна;

Oleksiy Balabukha – Candidate of Technical Sciences, Ivan Kozhedub Kharkiv National Air Force University, Doctoral Student, Kharkiv, Ukraine;

e-mail:alex15054444@gmail.com

ORCID ID: http://orcid.org/0000-0002-5263-9485

EVALUATION OF THE EFFECTIVENESS OF THE ENCHANCED GREEDY AND DELTA-DEBUGGING TEST CASE OPTIMIZATION ALGORITHMS FOR C++ LIBRARIES

Efficient optimization of test cases (TCs) is a necessary condition for improving the efficiency of testing C++ libraries. **The subject of the** research is test case optimization algorithms for C++ libraries. **The purpose of** this work is to improve the efficiency of C++ libraries testing by enhancing classical algorithms for greedy TC optimization and delta-debugging TC minimization. **Goals.** To improve the greedy TC optimization algorithm and eliminate its determinism, ensuring effective TC compression while preserving branch code coverage. To improve the delta-debugging TC minimization algorithm under conditions where redundant actions are sparsely distributed within TCs. To evaluate the effectiveness of the proposed algorithms in comparison with the classical algorithms. **Methods.** The study applies a greedy TC optimization algorithm, a delta-debugging TC minimization algorithm, mathematical modeling, and statistical analysis methods. The effectiveness of the improved algorithms is evaluated based on statistical analysis of 100 simulation runs for each algorithm on two open-source C++ libraries of different structural complexity. **Results.** The evaluation results indicate that the improved algorithms provide preservation (and possible increase) of branch coverage with coverage retention coefficient of up to 1.058, increase the TC compression ratio up to 0.86, and reduce the TS execution time by 1.5–2.5× compared to the classical algorithms. **Conclusions.** The improved algorithms significantly reduce the testing time of C++ libraries without loss of branch coverage. An improved greedy TC optimization algorithm is proposed, in which the selection of TC actions accounts for their future utility estimated by the expected increase in branch coverage. This approach removes the determinism of the classical greedy algorithm, increases the informativeness of TCs, and provides a balance between preserving branch coverage and TC compression. An improved delta-debugging algorithm is proposed that performs group removal of non-unique TC actions according to their contribution to branch coverage, which makes it possible to substantially reduce the length of TCs without losing testing effectiveness.

Keywords: software; test case; algorithm; optimization; coverage; time consumption; fault.

Бібліографічні описи / Bibliographic descriptions

Коломійцев О. В., Гулевич М. В., Дмітрієв О. М., Левченко А. О., Балабуха О. С. Оцінювання ефективності вдосконалених жадібного й дельта-дебагінг алгоритмів оптимізації тестових сценаріїв для С++ бібліотек. *Сучасний стан наукових досліджень та технологій в промисловості*. 2026. № 1 (35). С. 39–54. DOI: https://doi.org/10.30837/2522-9818.2026.1.039

Kolomiitsev, O., Hulevych, M., Dmitriiev, O., Levchenko, A., Balabukha, O. (2026), "Evaluation of the effecAtiveness of the enchanced greedyand delta-debugging test case optimization algorithms for C++ libraries", *Innovative Technologies and Scientific Solutions for Industries*, No. 1 (35), P. 39–54. DOI: https://doi.org/10.30837/2522-9818.2026.1.039