https://doi.org/10.15407/intechsys.2026.04.112
УДК 004.415.538

К.О. КОХАН, аспірант,
Національний університет біоресурсів і природокористування України,
вул. Героїв Оборони, 15, Київ, 03041, Україна
https://orcid.org/0009-0002-8878-8527
kokhan.kyrylo@gmail.com

ПРОДУКТИВНІСТЬ ТЕХНОЛОГІЇ ОПТИМАЛЬНИХ КОНФІГУРАЦІЙ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ БАГАТОКОМПОНЕНТНИХ СИСТЕМ

У статті досліджено продуктивність та обчислювальну стабільність технології оптимізації тестових конфігурацій багатокомпонентних систем. Технологію реалізовано як проміжне програмне забезпечення з інтерфейсом командного рядка (CLI Middleware) на базі середовища Node.js із застосуванням комбінаторних методів (Pairwise Testing, Orthogonal Arrays) та еволюційних обчислень (генетичних алгоритмів). Для усунення ризиків блокування головного циклу (Event Loop) впроваджено суворо детерміновані параметри еволюційного пошуку (30 поколінь, популяція 50). Експериментальне еталонне тестування підтвердило здатність системи стискати простір конфігурацій на 70,0–86,0% при математично доведеному збереженні 100% покриття попарних взаємодій (за допомогою розробленого програмного модуля валідації покриття). Зафіксований час виконання обчислювального ядра становить від 6 до 12 мс, що робить розробку ідеально придатною для синхронних конвеєрів безперервної інтеграції та розгортання (CI/CD).

Ключові слова: *оптимальні конфігурації, CLI Middleware, Node.js, Pairwise Testing, генетичні алгоритми, CI/CD, математичні моделі.*

Вступ

Масштабний перехід індустрії програмного забезпечення до розподілених мікросервісних та хмарних (*cloud-native*) архітектур суттєво підвищив гнучкість та обчислювальну потужність інформаційних систем. Водночас поліглотне збереження даних і технологічна гете-

Цитування: Кохан К.О. Продуктивність технології оптимальних конфігурацій для автоматизованого тестування багатокомпонентних систем. I*nformation Technologies and Systems*, Київ, 2026, Том 4 (10), 112–119. https://doi.org/10.15407/intechsys.2026.04.112

© Publisher PH "Akademperiodyka" of the NAS of Ukraine, 2025. This is an Open Access article under the CC BY-NC-ND 4.0 license (https://creativecommons.org/licenses/by-nc-nd/4.0/

рогенність призвели до виникнення явища «дрейфу середовищ» (*environment drift*). Забезпечення якості таких екосистем вимагає верифікації кодової бази на десятках комбінацій інфраструктурних параметрів, що охоплюють різні версії фронтенд-фреймворків, бекенд-рушіїв, систем керування базами даних та веб-браузерів [1–4].

Окреслення проблеми

В умовах сучасного розроблення програмного забезпечення на основі мікросервісних та хмарних архітектур верифікація якості вимагає тестування систем на гетерогенних конфігураційних матрицях. Повний комбінаторний перебір всіх можливих станів призводить до явища «комбінаторного вибуху», коли навіть для помірної кількості параметрів кількість конфігурацій сягає тисяч запусків. Виконання такого обсягу тестів у конвеєрах безперервної інтеграції та розгортання (*CI/CD*) стикається з нездоланними інфраструктурними й фінансовими обмеженнями: вичерпанням пулів з'єднань (*Connection Pool exhaustion*), перевищенням лімітів зовнішніх *API* (*rate limiting*) та критичним зростанням витрат на хмарні обчислення (*FinOps*) [3, 5–6]. Основна науково-технічна суперечність полягає в тому, що наявні еволюційно та комбінаторні алгоритми оптимізації просторів тестування розроблені для автономного виконання і не враховують обмежень середовищ реального часу *CI/CD*. Зокрема, імплементація обчислювально-містких генетичних алгоритмів або алгоритмів перебору в однопотоковому середовищі *Node.js* генерує високий ризик блокування головного циклу (*Event Loop*). Це призводить до відмови всього процесу інтеграції. Отже, виникає гостра потреба у розробці адаптивного механізму маршрутизації та детермінованих методів стиснення, які б гарантували високе покриття взаємодій за мінімального часу виконання (на рівні мілісекунд) без блокування ресурсів *CI/CD*.

Мета

Метою дослідження є проведення огляду продуктивності, експериментальне еталонне тестування та перевірка обчислювальної стабільності розробленого прототипу інформаційної технології (у форматі *CLI Middleware*) для вибору оптимальних тестових конфігурацій багатокомпонентних систем в умовах, наближених до промислових.

Хоча в академічному середовищі запропоновано чимало еволюційних алгоритмів для стиснення тестових просторів, вони переважно ігнорують реальні обмеження середовищ виконання. Наприклад, імплементація обчислювально важких генетичних алгоритмів (ГА) в асинхронному однопотоковому середовищі *Node.js* генерує високий ризик блокування головного циклу (*Event Loop*). Блокування головного циклу під час синхронного етапу конвеєра *CI/CD* є критичним архітектурним дефектом, що призводить до відмови всього процесу інтеграції [7].

Матеріали та методи

Фундаментальною архітектурною аксіомою розробленої системи є її функціонування виключно у форматі *CLI Middleware*, що ініціалізується як синхронний обчислювальний крок у межах конвеєра *CI/CD* (наприклад, у середовищі *GitHub Actions або Jenkins*). Технологія свідомо уникає парадигми мікросервісу на базі *REST API* для внутрішньої маршрутизації. Це архітектурне рішення повністю усуває мережеві затримки (*Network Latency*) та нівелює кількість потенційних точок відмови (*Points of Failure*) під час конвеєрного виконання.

Обчислювальний процес оркеструється модульною архітектурою, яка об'єднує підсистему управління життєвим циклом даних, модуль інтелектуальної адаптивної маршрутизації (селектор) та модуль алгоритмічної імплементації методів оптимізації.

Математично повний простір можливих тестових конфігурацій S_{full} формується як декартовий добуток розмірностей усіх параметрів системи. Такий підхід є базовою моделлю сучасного комбінаторного тестування та використовується для побудови компактних тестових наборів у промислових системах [3, 6]:

$$S_{full} = \Pi_{i=1}^{k} Vi, \tag{1}$$

де k — кількість інфраструктурних параметрів, а Vi — кількість можливих значень (варіативність) для i-го параметра.

Логіка функціонування підсистеми інтелектуальної маршрутизації (Рис. 1) реалізує багаторівневий алгоритм прийняття рішень на основі аналізу топології простору S_{full}:

1. Аналіз вхідного простору: на першому кроці обчислюється загальна кількість конфігурацій $N = |S_{full}|$ та середня варіативність параметрів $V = Avg(V_i)$ [8–9].

2. Маршрутизація за розмірністю (N):

• малий простір ($N \leq 100$): передається безпосередньо модулю *Pairwise Testing* (*Greedy*), оскільки для малих матриць жадібні алгоритми забезпечують миттєву генерацію оптимального покриття за час ≤ 10 мс;

• середній простір ($100 < N \leq 500$): вибір алгоритму додатково аналізує фактор варіативності V. Якщо $V \leq 2{,}5$ (низька варіативність, параметри мають збалансовану кількість значень), застосовуються ортогональні масиви (*Orthogonal Arrays*), які дають математично збалансоване рівномірне покриття. Якщо $V > 2{,}5$ (висока дисперсія розмірностей), система маршрутизує задачу до *Pairwise Testing*, який краще адаптується до незбалансованих факторів;

• надвеликий простір ($N > 500$): система автоматично підключає детермінований генетичний алгоритм, оскільки комбінаторні методи на великих матрицях втрачають обчислювальну ефективність, а (ГА) дає змогу знайти квазіоптимальне рішення за фіксовану кількість поколінь (*Gen* = 30, *PopSize* = 50).

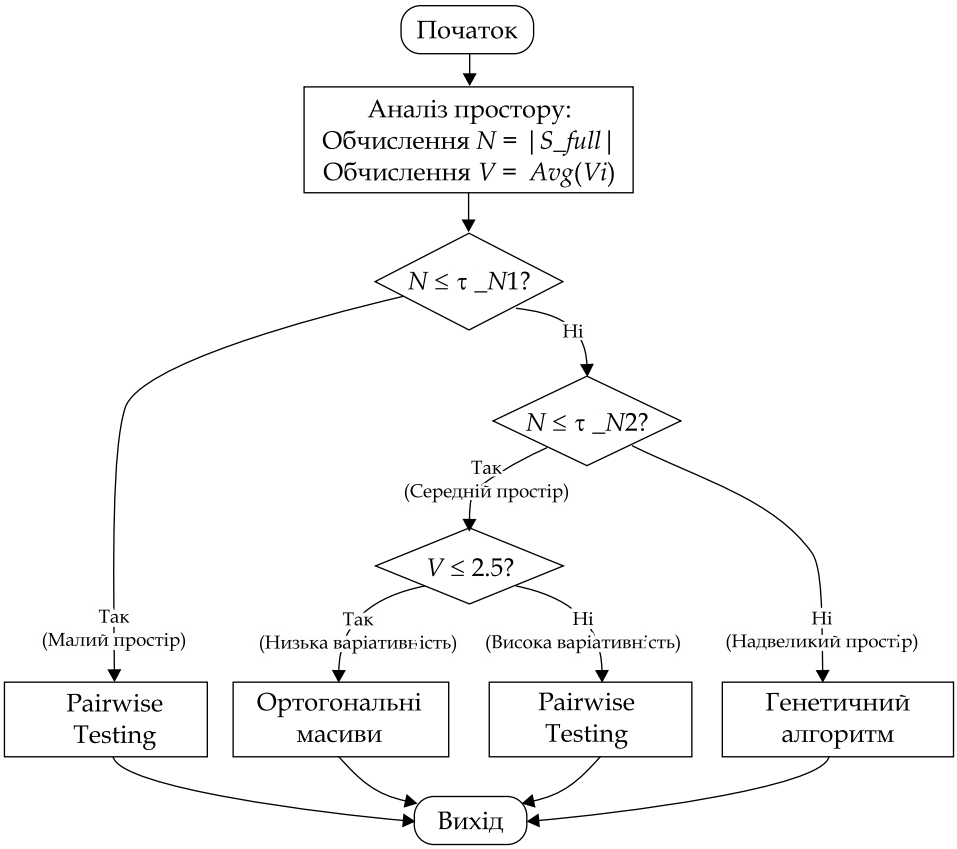


Рис. 1. Блок-схема адаптивного вибору методів оптимізації підсистемою маршрутизації(на рис. 1 англ. літери прокурсивити)

Для абсолютного унеможливлення блокування головного циклу під час роботи генетичного алгоритму, на еволюційний цикл накладено суворі інфраструктурні обмеження:

$$Gen = 30; \ PopSize = 50. \tag{2}$$

Механізм раннього зупинення (*Early Stopping*) було навмисно відключено. Цей крок забезпечує константну часову складність $O(1)$ відносно кількості ітерацій i гарантує суворо передбачуваний час виконання скрипту, незалежно від наявності аномалій у вхідних даних.

Генетичний алгоритм оперує популяцією хромосом (масивів конфігурацій). Використовуються такі оператори:

- селекція: турнірний відбір з динамічним розміром турніру;
- схрещування (*Crossover*): одноточковий кросовер зі зсувом для хромосом різної довжини;
- мутація: точкове заміщення з імовірністю $p_m = 0{,}2$ та обов'язковою внутрішньою дедуплікацією генів (конфігурацій).

З метою балансування навантаження на обчислювальні вузли (*workers*) алгоритм також використовує функцію аугментації, яка за-

побігає створенню оптимальних матриць S_{opt} розміром менше мінімального порогу N_{min} = 5, дублюючи релевантні конфігурації для ефективного завантаження доступних паралельних вузлів кластера.

Математична валідація повноти покриття дефектів взаємодії суворо ізольована в розробленому програмному модулі валідації покриття. Цей модуль рекурсивно перевіряє наявність усіх згенерованих попарних (2-*wise*) комбінацій у результуючій матриці S_{opt}, гарантуючи якість стиснення.

Для кількісного оцінювання та порівняльного аналізу ефективності розробленої технології порівняно з наявними підходами застосовано 7-факторний інтегральний критерій ефективності $I(j)$, формалізований та апробований авторами у попередніх працях [5, 7]. Математично критерій є адитивною згорткою нормалізованих факторів:

$$I(j) = \sum_{i=1}^{7} W_i \cdot P_{ij} \to \max \in [0, 1], \quad (3)$$

де $I(j)$ — ідентифікатор аналізованого методу оптимізації; $P_{ij} \in [0, 1]$ — нормалізоване значення i-гоо параметра ефективності для j-го методу; W_i — ваговий коефіцієнт значущості j-го критерію ($\sum_{i=1}^{7} W_i = 1$), обчислений на основі аналізу наукових праць та експертного оцінювання [5].

1. Кількість тестів (W_1 = 0,20): оцінка ступеня скорочення підмножини конфігурацій порівняно з повним декартовим простором S_{full};

2. покриття критичних сценаріїв (W_2 = 0,25): повнота покриття двофакторних (2-*wise*) взаємодій компонентів (максимальний пріоритет надійності);

3. адаптивність до змін (W_3 = 0,15): здатність методу динамічно пристосовуватися до зміни вхідної варіативності параметрів ІС;

4. використання історичних даних (W_4 = 0,10): рівень залежності алгоритму від наявності накопиченої статистики про минулі запуски (відсутність проблеми «холодного старту»);

5. інтеграція з *CI/CD* (W_5 = 0,15): безшовність впровадження ядра у конвеєри безперервної інтеграції та розгортання (*GitHub Actions, Jenkins*);

6. обчислювальні ресурси (W_6 = 0,10 рівень витрат процесорного часу та оперативної пам'яті на виконання самого алгоритму оптимізації;

7. складність впровадження (W_7 = 0,05): трудомісткість налаштування та конфігурування програмного прототипу.

Результати

Експериментальне тестування (бенчмаркування) проводилося на двох типових синтетичних наборах параметрів, що достовірно імітують багатокомпонентні середовища:

• Набір 1 (низька складність): 5 параметрів із розмірностями (2, 3, 2, 3, 3), що формують S_{full} = 108 конфігурацій.

Таблиця 1. **Емпіричні результати стиснення простору конфігурацій (Бенчмаркування)**

Тестовий набір	Початковий обсяг S_{fill}	Метод оптимізації (Селектор)	Оптимізований обсяг S_{opt}	Коефіцієнт стиснення (%)
Suite 1	108 конфігурацій	*Pairwise Testing* (*Greedy*)	15 конфігурацій	86,11%
Suite 2	243 конфігурації	*Integrated Method* (*Pairwise* + *GA*)	34 конфігурації	86,01%
У середньому	≈100-300	Адаптивна маршрутизація	12 – 35	70,0 – 86,0%

• Набір 2 (середня складність): 5 параметрів із розмірностями (3, 3, 3, 3, 3), що формують S_{full} = 243 конфігурації.

Усі обчислення та метрики продуктивності автоматично агрегувалися у вихідних файлах звітів та метрик. Ключовим показником успішності є коефіцієнт алгоритмічного просторового стиснення, який прямо пропорційно зменшує інфраструктурні витрати. Емпіричні результати подано в табл. 1.

Як засвідчують експериментальні дані, розроблений *CLI Middleware* стабільно зменшує обсяг тестових запусків на 70,0–86,0 %. Це повністю ліквідує проблему вичерпання пулів з'єднань, оскільки кількість необхідних паралельних з'єднань з базами даних скорочується із сотень до безпечного ліміту в 15–35 активних вузлів.

Критично важливим є те, що журнали виконання модуля перевірки покриття підтвердили: для обох тестових наборів згенеровані стиснуті матриці гарантують 100% покриття всіх можливих попарних комбінацій (2-*wise coverage*), що відповідає сучасним підходам комбінаторного тестування, згідно з якими більшість дефектів спричиняються взаємодією невеликої кількості параметрів, а ефективне покриття 2–6-*way* взаємодій забезпечує високу надійність систем при істотному скороченні кількості тестів [3–4, 8–9]. Це є суворим математичним доказом того, що відсікання 86 % простору конфігурацій відбулося без жодного зниження ймовірності виявлення дефектів взаємодії.

Зафіксований час виконання обчислювального алгоритмічного ядра в середовищі *Node.js* знаходився у жорстких межах від 6 до 12 мс. Цей показник остаточно підтверджує успішне вирішення проблеми блокування головного циклу і гарантує відсутність операційних вузьких місць. На основі проведених вимірювань, емпіричний інтегральний показник ефективності для запропонованого інтегрованого підходу склав: $I(j)$ = 0,9725.

Висновки

У межах дослідження доведено критичну важливість оптимізації тестових просторів та експериментально підтверджено високу продуктивність розробленої інформаційної технології. Архітектурне проєктування системи у вигляді *CLI Middleware* на платформі *Node.js* забезпечує безпрецедентну швидкість обчислень на рівні мілісекунд (6–12 мс), повністю виключаючи ризики блокування головного циклу завдяки суворій детермінації параметрів еволюційного пошуку (відсутність раннього зупинення, рівно 30 поколінь, популяція 50).

Встановлено, що запропонований інтелектуальний механізм маршрутизації здатен автоматично стискати декартові простори конфігурацій (розмірністю 108 та 243) на 70,0–86,0 %. Водночас модуль перевірки покриття засвідчив беззаперечне збереження стовідсоткового 2-*wise* покриття. Досягнення емпіричного інтегрального показника $I(j) = 0,9725$ підтверджує абсолютну готовність розробленої технології до промислової інтеграції в корпоративні конвеєри *CI/CD*, що ефективно розв'язує бюджетні обмеження *FinOps* та ліміти інфраструктурних потужностей без компромісів щодо якості тестування.

ПОВІДОМЛЕННЯ

Внесок авторів. *Кохан К. О. – концептуалізація, розробка програмного забезпечення, проведення експериментів, збір даних, перевірка результатів, написання тексту.*

Застосування ШІ та інших інструментів. *Автор заявляє, що інструменти штучного інтелекту не використовувалися при написанні статті.*

Конфлікт інтересів. *Автор заявляє про відсутність будь-яких фінансових чи нефінансових конфліктів інтересів, які могли б вплинути на результати цього дослідження.*

Фінансування. *Автор заявляє, що під час підготовки цього рукописи не отримував жодних коштів, грантів чи іншої підтримки.*

ЛІТЕРАТУРА / REFERENCES

1. Grindal M., Offutt J., Andler S.F. Combination Testing Strategies: A Survey. *Software Testing, Verification and Reliability*, 2005, Vol. 15 (3), 167–199. https://doi.org/10.1002/stvr.319
2. Nie C., Leung H. A Survey of Combinatorial Testing. *ACM Computing Surveys*, 2011, Vol. 43 (2), 1–29. https://doi.org/10.1145/1883612.1883618
3. Raunak M.S., Kuhn D.R., Kacker R.N., Lei Y. Combinatorial Testing for Building Reliable Systems. *IEEE Reliability Magazine,* 2024, Vol. 1 (1), 15–19. https://doi.org/10.1109/MRL.2024.3355874
4. Baumann C., Koroglu Y., Wotawa F. On the Impact of Input Models on the Fault Detection Capabilities of Combinatorial Testing. *SN Computer Science*, 2024, Vol. 5 (7), Article 821. DOI: https://doi.org/10.1007/s42979-024-03134-3
5. Kokhan K.O., Tkachenko O.M. Review and proposal for optimization of optimal configurations for automated testing of multi-component information systems. *Information Technologies and Society*, 2025, Vol. 3 (18), 83–87. https://doi.org/10.32689/maup.it.2025.3.11
6. Kuhn R., Kacker R., Lei Y. Introduction to Combinatorial Testing. *CRC Press*, Boca Raton, 2013. https://doi.org/10.1201/b14957
7. Kokhan K.O., Tkachenko O.M. Implementation and testing of the configuration optimization module for automated testing of multi-component information systems.

Computer-integrated technologies: education, science, production, 2025, Vol. 61, 18–23. https://doi.org/10.36910/6775-2524-0560-2025-61-03
8. Lei Y., Tai K. C. In-Parameter-Order: A Test Generation Strategy for Pairwise Testing. *3rd IEEE International High-Assurance Systems Engineering Symposium*, 1998, 254–261. https://doi.org/10.1109/HASE.1998.731623
9. Mandl R. Orthogonal Latin Squares: A Tool for the Design of Experiments in Testing. *Communications of the ACM*, 1985, Vol. 28 (10), 1054–1058. https://doi.org/10.1145/4372.4375

Отримано/Received 01.07.2026
Прийнято/Accepted 09.09.2026
Опубліковано/Published 23.10.2026

K.O. KOKHAN, PhD Student,
National University of Life and Environmental Sciences of Ukraine,
15, Heroiv Oborony St., Kyiv, 03041, Ukraine
https://orcid.org/0009-0002-8878-8527
kokhan.kyrylo@gmail.com

PRODUCTIVITY OF TECHNOLOGY OPTIMUM CONFIGURATIONS FOR AUTOMATED TESTING OF RICH COMPONENT SYSTEMS

Introduction. The transition of the software industry to distributed microservice and cloud-native architectures has significantly increased the flexibility of information systems, but has led to the phenomenon of "environment drift." Exhaustive testing of such ecosystems causes a "combinatorial explosion," which, under continuous integration and continuous deployment (CI/CD) pipelines, leads to connection pool exhaustion and critical budget overruns on cloud computing (FinOps), making complete enumeration impossible.

Objective. Conducting a performance review, experimental benchmarking, and verifying the computational stability of the developed prototype of information technology (in the format of CLI Middleware) for selecting optimal test configurations of multi-component systems under conditions close to industrial ones.

Methods. An information technology has been developed in the form of CLI Middleware based on Node.js using combinatorial methods (Pairwise Testing, Orthogonal Arrays) and evolutionary computations (genetic algorithms). To completely prevent the blocking of the main execution cycle (Event Loop), strictly deterministic evolutionary search parameters have been implemented (30 generations, population of 50) and the early stopping mechanism (Early Stopping) has been disabled.

Results. Experimental benchmarking on synthetic test suites (108 and 243 configurations) confirmed the system's ability to automatically compress the configuration space by 70.0–86.0%. An isolated mathematical validation module proved the preservation of 100% coverage of pairwise (2-wise) interactions. The recorded execution time of the computational core is 6–12 ms, making it ideally suited for use as a synchronous step in CI/CD pipelines.

Conclusions. The effectiveness of the developed information technology has been proven, successfully solving the "combinatorial explosion" problem and eliminating the risks of blocking the main cycle. Achieving an integral efficiency metric of 0.9725 confirms the absolute readiness of the CLI Middleware architecture for industrial integration into corporate pipelines, allowing significant optimization of infrastructure costs without compromising the quality of interaction defect detection.

Keywords: *optimal configurations, CLI Middleware, Node.js, Pairwise Testing, genetic algorithms, CI/CD, mathematical models.*