https://doi.org/10.31891/2219-9365-2023-74-27
УДК 004.051:004.052.2:004.052.3

ФОРКУН Юрій
Хмельницький національний університет
https://orcid.org/0000-0002-7906-4191
forkun@ridne.net
ФОРКУН Ірина
Хмельницький національний університет
https://orcid.org/0000-0002-4588-6349
ivforkun@gmail.com
ЯШИНА Оксана
Хмельницький національний університет
https://orcid.org/0000-0001-7816-1662
ksusha.ja@gmail.com
ПРАВОРСЬКА Наталія
Хмельницький національний університет
https://orcid.org/0000-0001-6001-3311
margana2000007@gmail.com

АРХІТЕКТУРНІ МЕТОДИ ОПТИМІЗАЦІЇ ШВИДКОДІЇ ТА ВІДМОВОСТІЙКОСТІ ПРОГРАМНИХ ЗАСТОСУНКІВ

В роботі наведено результати досліджень шляхів та архітектурних методів оптимізації програмних застосунків з метою покращення їх швидкодії та відмовостійкості, їх вплив на якість програмного забезпечення та проаналізовано їх переваги та недоліки.

Ключові слова: архітектура, швидкодія, відмовостійкість, дані, програмний код, програмний застосунок, бази даних, методи оптимізації програмних продуктів

FORKUN Yurii, FORKUN Iryna, YASHINA Oksana, PRAVORSKA Nataliia
Khmelnytskyi National University

ARCHITECTURAL METHODS OF OPTIMIZING THE SPEED AND FAULT TOLERANCE OF SOFTWARE APPLICATIONS

One of the most important aspects of the quality of software products and systems, both today and against the background of the entire violation of software technologies, both devices and productivity and speed of coding. The composition of programs to process these various volumes in the process of working for the set goal for the first time, when one of the most important goals was without powering some software systems. In modern realities, the need for safe, accepted high-speed coding systems is greater than evil. The reason for this, on the one hand, is the increase in competition in the market of software products, as well as the greater importance of such categories as big data.

In general, refactoring and reengineering of software applications are mostly used to optimize the software code. However, they are used mainly on already developed and working applications. The main method of optimization in the development of software applications is the use of correct architectural solutions and methods of choosing and applying the architecture of software applications at the stage of their modeling and design.

That is, in the modern realities of the field of software engineering, there is a need to ensure the optimization of application software codes, taking into account their possible architectural solutions.

The purpose of the work is the analysis of possible solutions for code optimization and the assessment of their advantages and possible disadvantages when applying the methods of architectural solutions of software applications at the stage of their modeling and design.

Keywords: architecture, speed, fault tolerance, data, software code, software application, databases, optimization methods of software products.

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

Одним із найбільш важливих аспектів якості програмних продуктів та систем, як на сьогоднішній день, так і на протязі всього існування програмної інженерії як галузі є їх продуктивність та швидкодія. Здатність програми обробляти дані різних об'ємів в прийнятний для поставленої цілі час завжди була однією з найголовніших цілей забезпечення якості програмних систем. В сучасних реаліях потреба в забезпеченні прийнятної швидкодії системи лише зросла. Причиною цього є, з одного боку, збільшення конкуренції на ринку програмних продуктів, а з іншого зростання важливості такої категорії як великі дані.

Великі дані в сучасному їх розумінні – це великі масиви даних, які виникають в наслідок використання людьми мережі Інтернет. Їх обробка та розуміння можливі лише з допомогою спеціалізованих інструментів та методів. Роль великих даних в сучасному світі важко переоцінити. Дослідження показують,

що використання та обробка великих даних допомагає зрозуміти тенденції та настрої як людського суспільства в цілому, так і окремих його частин [1][2]. Обробка великих даних досить важлива для систем з підтримкою штучного інтелекту, ед вимоги до оптимізації програмного коду є досить значними та висувають свої певні вимоги . Це, в свою чергу, може знайти застосування майже в всіх сферах людської діяльності – від бізнесу, до соціологічних та політичних сфер. Саме для успішної обробки та аналізу великих обсягів даних добре оптимізований код не просто є бажаним, він є необхідним.

Проте, оптимізація коду потребується не лише в сфері роботи з великими даними. Сучасні програмні системи та застосунки також отримують ряд переваг при оптимізації коду. Одним з найбільш очевидних бенефітів від оптимізації коду є покращення відмовостійкості в ситуаціях пов'язаних з великими навантаженнями. В сучасних реаліях більшість коммерційних програм в тій чи іншій мірі зав'язані на мережі інтернет, що в свою чергу передбачає велику кількість користувачів. З кожним днем користувачів інтернету стає все більше, а відповідно зростає і число потенційних користувачів системи. І хоча хмарні технології також не стоять на місці, покладатись лише на потужність ресурсів сервера було б великою помилкою. Добре оптимізований код дозволяє забезпечити плавну роботу з великим навантаженням, і забезпечує певний запас потужності на випадок, якщо реальне навантаження на систему з певних причин перевищить очікувані.

Окрім того, оптимізація коду несе переваги не лише для користувачів системи. Однією з важливих переваг оптимізованого коду в довгостроковій перспективі є легкість підтримки. Оптимізований код поперше, не містить надлишкових компонентів, що спрощує розуміння системи, а по-друге оптимізований код сам по собі є легким для розуміння, оскільки зазвичай він не містить надзвичайно заплутаних структур та алгоритмів, які б були важкими для розуміння. Таким чином, забезпечується легкість підтримки та можливість залучення інших команд розробників у роботу над системою.

Загалом до оптимізації програмного коду застосовують здебільшого методи рефакторингу та реінженерію програмних застосунків. Однак вони застосовуються в основному на вже розроблених та працюючих застосунків. Для оптимізації програмного коду та обробки даних на даний час серед сучасних підходів методів оптимізації доцільно виділити метод заснований на правилах механізму спеціалізації програм, який забезпечує значну оптимізацію продуктивності коду обробки даних з поступовою типізацією[3]. Такий підхід дозволяє раннє виявлення помилок типу під час компіляції, який забезпечує безпеку статичних типів даних. У роботі [4] розглядається оптимізація коду як проблема оптимізації: від змішаного цілочисельного програмування до покращеного високопродуктивного рандомізованого GRASPподібного алгоритму, який має набагато кращий час виконання, ніж модель MIP.

В умовах сьогодення, для оптимізації програмного коду та програмнних застосунків все частіше застосовуються різні алгоритми та методи штучного інтелекту (ШІ). Так, зокрема, у роботі [5] подано використання математичної моделі алгоритмів ШІ для оптимізації комп'ютерного програмування та висвітлено велику важливість і практичну роль методів оптимізації математичних моделей і математичних алгоритмів на основі алгоритмів ШІ при оптимізації програмних застосунків. Результати такого підходу показують, що оптимізація на основі математичних моделей алгоритмів ШІ в деякій мірі може досить ефективно покращити організацію та ієрархію як коду, так і роботу самих застосунків.

Однак, основним з методів оптимізації при розробці програмних застосунків є застосування вірних архітектурних рішень та методів вибору і застосування архітектури програмних застосунків на етапі їх моделювання та проектування.

Тобто, у сучасних реаліях галузі програмної інженерії є потреба в забезпеченні оптимізації програмних кодів застосунків з урахуванням їх можливих архітектурних рішень.

Формулювання цілей статті

Метою роботи є аналіз можливих рішень по оптимізації коду програмних застосунків при обробці даних та оцінка їх переваг та можливих недоліків при застосуванні методів архітектурних рішень програмних застосунків на етапі їх моделювання та проектування.

Виклад основного матеріалу

Проведені нами дослідження показали, що загалом існує декілька основних цілей, для яких може здійснюватися оптимізація програмного коду:

- покращення швидкодії;
- оптимізація обробки даних;
- покращення відмовостійкості;
- оптимізація об'єму коду.

Найпоширенішим процесом оптимізації є оптимізація продуктивності, зокрема його швидкодії. Продуктивність важлива майже у всіх програмних системах. У системах реального часу вона є критично важливою. Це пов'язано з тим, що вся система базується на забезпеченні стабільної роботи програмного

застосунку протягом чітко визначеного часового інтервалу. У більш традиційних додатках, призначених для звичайних користувачів, продуктивність не настільки важлива, як у системах реального часу, але в сучасному контексті в користувацьких додатках інтерфейс користувача відіграє особливо важливу роль. Саме швидкодія є тим основним критерієм за яким відбувається конкуренція між подібними програмними застосунками.

Існує кілька методів підвищення продуктивності. Найпоширеніший - це розбиття однієї великої команди на декілька менших команд, щоб зменшити загальний час виконання. До прикладу, може виникнути ситуація, розбиття одного великого запиту до бази даних на кілька менших запитів, в результаті якого буде отримано той самий набір даних приведе до того, що додаток буде працювати значно швидше зберігаючи усю свою функціональність.

Код також може містити надлишкові команди. До них відносяться змінні, яким присвоюються значення, що ніде не використовуються, неправильні логічні перевірки та непотрібні запити до бази даних. Вважається доброю практикою підтримувати код у чистоті, оскільки такі випадки не тільки сповільнюють роботу програми, але й погіршують розуміння програмного коду та відволікають увагу від більш важливих компонентів застосунку.

При виконанні такого рефакторингу слід також пам'ятати про якість програмного коду. Якість коду включає в себе не тільки функціональні властивості, але й такі, як читабельність, зрозумілість і складність. Хорошим способом оцінити такі властивості є використання типових метрик кодування, які надають числові характеристики модулів програмного застосунку.

Іншим методом оптимізації є прогнозування результатів. Можна наперед передбачити подальші кроки користувача і, відповідно, виконати подальше обчислення до того моменту, як воно знадобиться. Іншим застосуванням цих методів є побудова таблиць результатів часто використовуваних функцій. Це стосується, наприклад, побудови таблиці пошуку частих значень функції, яка обчислює математичні функції. Перед виконанням обчислення функція порівнює аргументи з таблицею, коли відповідні дані знайдено, функція одразу поверне значення без будь-яких подальших складних обчислень.

Також при цьому слід врахувати методи покращення часу відгуку інтерфейсу. На відміну від традиційних методів оптимізації продуктивності, метою є не безпосереднє прискорення операцій, а зменшення часу бездіяльності користувацького інтерфейсу. Ці методи призначені для створення певної видимості високої продуктивності, однак приводять до покращення користувацького досвіду роботи з програмним застосунком. Створення чуйного інтерфейсу з коротким часом бездіяльності значно покращує простоту використання і звичність інтерфейсу, що є однією з головних вимог до інтерфейсу користувача програмної системи.

Одним із перши запропонованих методів підвищення продуктивності та дієвим зараз, є вставка коду на низькорівневих мовах програмування, наприклад на асемблері, коли команди виконуються безпосередньо, без обробки компілятором. Однак такий підхід також має певні недоліки, зокрема недоліком такого підходу є те, що отриманий код є більш складним і неоднорідним. В результаті код може стати складнішим для розуміння. Тому, з цієї причини такий підхід часто використовується лише у крайньому випадку.

Застосування оптимізації коду для підвищення стабільності гарантує, що застосунок працюватиме стабільно за різних умов. Тут головна увага при оптимізації в цій області приділяється обробці виняткових випадків і забезпеченню стабільної роботи програми під навантаженням. Крім того, якщо у програмі використовуються певні зовнішні компоненти, такі як зовнішні API та бази даних, то варто розглянути можливість функціонування програми, навіть якщо ці компоненти недоступні за певних умов.

Продовженням оптимізації коду та окремим випадком оптимізації є оптимізація розміру коду. Вона в першу чергу спрямована на застосунки, що працюють на різноманітних мобільних пристроях з вкрай обмеженим обсягом внутрішньої пам'яті. Такий процес оптимізації часто негативно впливає на стабільність та відмовостійкість застосунку, але в таких випадках такі дії є необхідними для загального використання .

Отже, в результаті дослідження ми отримали три основні напрямки оптимізації програмних застосунків: оптимізація продуктивності, оптимізація відмовостійкості та оптимізація розміру коду. Слід зазначити, що, наприклад оптимізація розміру коду може мати негативний вплив на продуктивність та відмовостійкість, тоді як оптимізація продуктивності може призвести до зниження стабільності тощо. Тому при розробці програмного проекту слід визначити пріоритетність кожного з цих напрямків.

Визначати пріоритетність для більшої наочності будемо у вигляді діаграми (Рисунок 1). На цій діаграмі порівнюються три визначені напрямки оптимізації за п'ятибальною шкалою, де 5 - найвищий пріоритет напрямку, а 0 - відсутність пріоритету. Тобто, можна чітко вказати важливість процесу оптимізації в кожному напрямку і відповідно спланувати проект. Щоб уникнути ситуації, коли всі пріоритети є рівними і, відповідно, жоден напрямок оптимізації не є пріоритетним, максимальна сума пріоритетів повинна бути обмежена значенням, відмінним від 3. Наприклад, доцільно встановити максимальне значення загальної суми пріоритетів на рівні 6, щоб, якщо один напрямок має найвищий пріоритет, інші напрямки могли мати лише найнижчий пріоритет.

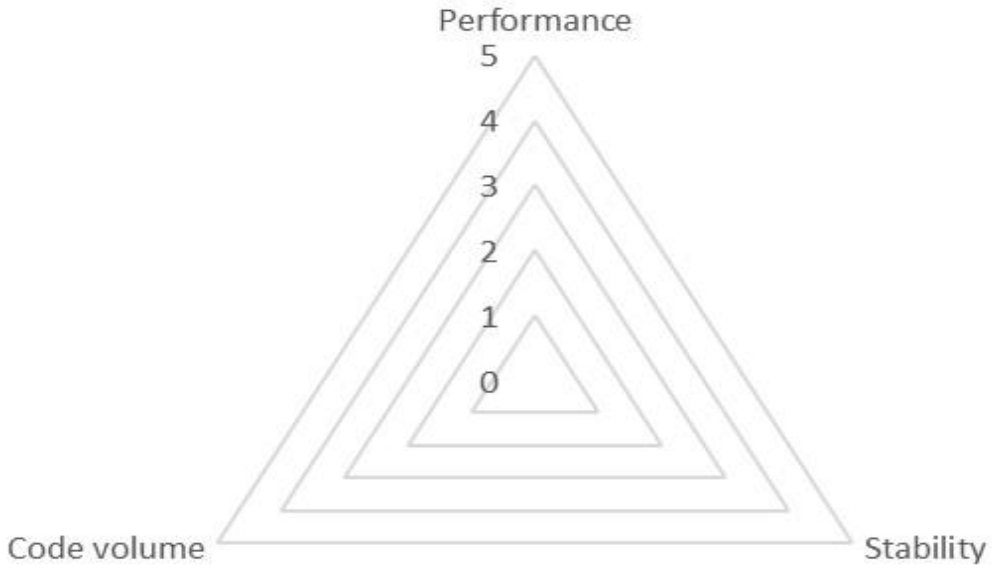

Рис. 1. Діаграма пріорітетів оптимізації

Дана діаграма приймає різні значення для різних проектів, можна визначити основні типи проектів та їхні пріоритетні характеристики, залежно від вимог, які висуває програмна система, що розробляється. Найпоширенішим типом проекту є загальний користувацький додаток, призначений для звичайних користувачів. Користувацькі додатки зазвичай мають графічний інтерфейс і тому вони вимагають уваги до відповідного рівня продуктивності для забезпечення продуктивної роботи користувача. Крім того, важливо забезпечити стабільність. Це пов'язано з тим, що більшість користувачів не мають відповідного досвіду і навичок для вирішення складних проблем, які можуть виникнути в процесі роботи з додатком. Оптимізація розміру коду в таких випадках не є особливо важливою, оскільки такі системи зазвичай встановлюються на потужні пристрої з високими показниками продуктивності. Даний випадок зображений на діаграмі пріоритетів оптимізації, яку показано на рисунку 2.

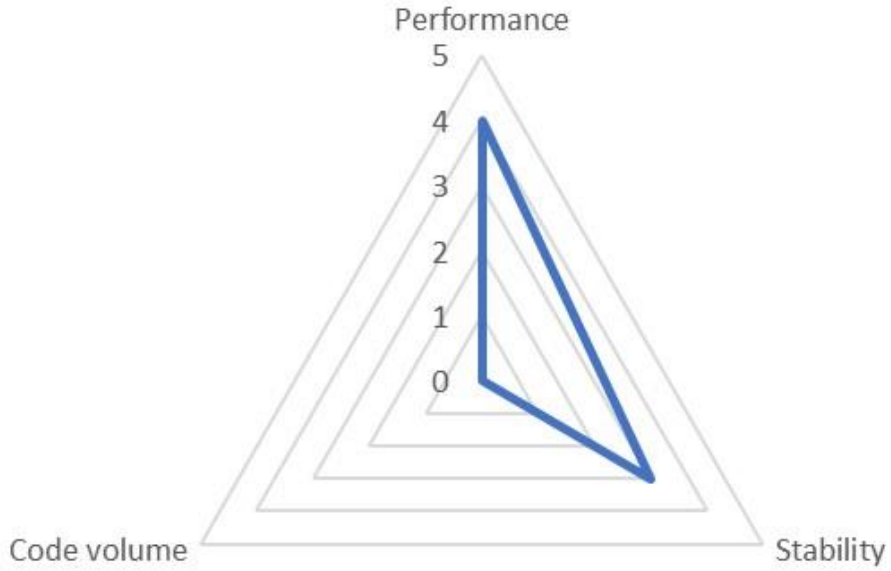

Рис. 2. Діаграма пріорітетів оптимізації для користувацьких додатків

Наступним типом програмного проекту є система реального часу. Для даного типу оптимізація стабільності є високим пріоритетом. У таких випадках продуктивність важлива, але поява помилок, які зупиняють або серйозно погіршують роботу системи, зазвичай має набагато гірші наслідки. Тут оптимізація розміру коду також не рекомендується для такого типу систем, оскільки це може призвести до неприйнятної продуктивності та погіршення стабільності. Діаграма пріоритетів для систем реального часу набуде вигляду, як показано на рисунку 3.

ISSN 2219-9365

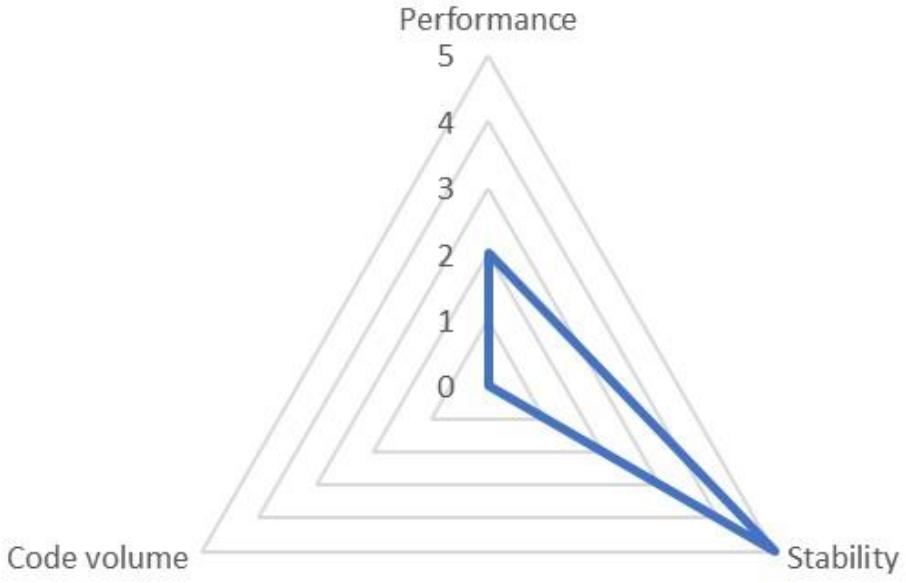

Рис. 3. Діаграма пріорітетів систем реального часу

Іншим типовим типом програмних застосунків є вбудовані системи. Для таких систем основним пріоритетом є показник зменшення кількості коду. Це пов'язано з тим, що такі системи зазвичай працюють з обмеженим об’ємом пам'яті. Оптимізація стабільності також важлива. Вбудовані системи не завжди здатні правильно відображати повідомлення про помилки, тому більшість виняткових випадків доводиться обробляти самій системі. У таких випадках продуктивність зазвичай менш критична, оскільки ці системи зазвичай виконують не досить складні обчислення, ніж додатки, які призначені для роботи у звичайних операційних системах на типових комп'ютерах. Тут також можливі випадки, коли потужність пристрою, на якому виконуються ці програми, може бути обмежена, тому у цьому випадку також слід враховувати оптимізацію продуктивності. На рисунку 4 наведено типовий приклад діаграми пріоритетів таких систем.

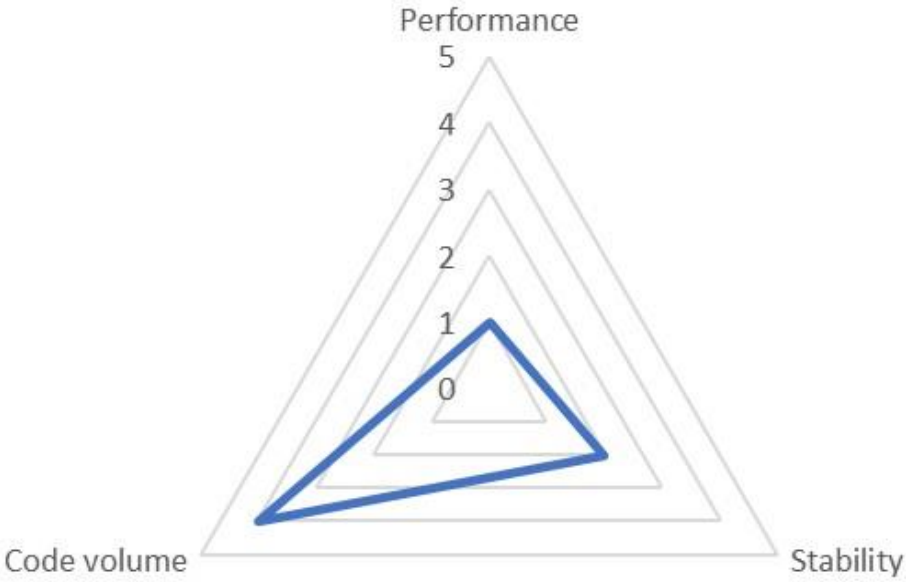

Рис. 4. Діаграма пріорітетів оптимізації вбудованих систем

Варто зазначити, що кожен проект може мати свої власні вимоги, які відрізняються від загального випадку, тому діаграму пріоритетів оптимізації слід створювати окремо для кожного проекту. Цю діаграму слід створювати на початку проекту, коли відомий тип системи та основні вимоги. На основі цієї діаграми розробники можуть розробляти та здійснювати рефакторинг відповідно до потреб проекту та приймати рішення щодо важливості тих чи інших характеристик програмного застосунку, якщо під час розробки виникають розбіжності між ними.

ISSN 2219-9365

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

Таким чином, в роботі було проведено аналіз архітектурних методів оптимізації швидкодії та відмовостійкості програмних застосунків. Було розглянуто основні архітектурні підходи методи та прийоми, які застосовуються для оптимізації коду для обробки даних з метою покращення його швидкодії та стабільності. До найбльш поширених та прийнятних методів оптимізації належать розбиття великих команд на менші та проведення рефакторингу з метою видалення надлишкових команд. Більш специфічні методи, такі як вставки низькорівневого коду в програму також мають місце, проте ними не слід зловживати, оскільки вони можуть нести негативні наслідки для читабельності та безпеки програми в цілому. Крім цього, також елементами оптимізації стабільності роботи програм є забезпечення обробки виключних випадків та передбачення можливості роботи системи при відмові зовнішніх компонент, якщо такі використовуються в програмному застосунку.

Було розроблено метод визначення пріорітетів оптимізації програмних проектів за допомогою побудови діаграм. Використовуючи ці діаграми в процесі розробки архітектури застосунку, рефакторингу та реінжинірингу можна значно точніше контролювати відповідність проекту заданим вимогам та забезпечувати коректну поведінку програмної системи відповідно до умов в яких вона функціонуватиме.

References

1. Profiling and optimization of Python-based social sciences applications on HPC systems by means of task and data parallelism. Lukasz Szustak, Marcin Lawenda, Sebastian Arming, Gregor Bankhamer, Christoph Schweimer, Robert Elsässer // Future Generation Computer Systems. Volume 148, 2023, Pages 623-635. https://doi.org/10.1016/j.future.2023.07.005
2. Кислова О. М. Великі дані в контексті дослідження проблем сучасного суспільства / О. М. Кислова // Вісник Харківського національного університету імені В.Н. Каразіна – 2019, №42
3. Rule-based program specialization to optimize gradually typed code. Francisco Ortin, Miguel Garcia, Seán McSweeney // Knowledge-Based Systems Volume 179, 1 September 2019, Pages 145-173. https://doi.org/10.1016/j.knosys.2019.05.013
4. Code design as an optimization problem: from mixed integer programming to an improved high performance Randomized GRASP like algorithm. José Barahona da Fonseca // Computer Aided Chemical Engineering, Volume 24, 2017, Pages 279-284. https://doi.org/10.1016/S1570-7946(07)80070-8
5. Optimization of computer programming based on mathematical models of artificial intelligence algorithms. Yuhui Zheng // Computers and Electrical Engineering. Volume 110, September 2023, 108834. https://doi.org/10.1016/j.compeleceng.2023.108834