УДК 004.9

МАТЕМАТИЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПОШУКУ КЛОНІВ КОДУ НА ОСНОВІ СЕМАНТИЧНИХ МЕТОДІВ

Мельник А.М.[1)], Лавор М.Ю.[2)], Романюк М.В.[3)], Тимчишин В.С.[4)]
Західноукраїнський національний університет
[1)]к.т.н., доцент; [2)]магістрант; [3,4)]аспірант

I. Постановка проблеми

Широке поширення вільного програмного забезпечення призвело до частого використання готових фрагментів вихідного коду при розробці нового програмного забезпечення (ПЗ). Розробники можуть використовувати як інтернет-ресурси, так і код, написаний ними самими або їх колегами. Згідно з результатами досліджень, програми можуть містити до двадцяти відсотків клонів коду (скопійованих фрагментів). Безконтрольне клонування може призвести до збільшення розміру початкового та бінарного коду програми, виникненню семантичних помилок, ускладнення підтримки ПЗ та ін. Відомо, що проекти FreeBSD і Linux містять сотні помилок, пов'язаних з клонуванням коду.

На сьогоднішній день не багато інструментів пошуку клонів коду, які забезпечували б необхідний рівень точності і масштабування до десятків мільйонів рядків вихідного коду, ні досить точних інструментів пошуку семантичних помилок в клонах. Тому розробка адекватних математичних і програмних засобів, що дозволяють вирішувати вказані проблеми є актуальною задачею.

II. Мета роботи

Метою дослідження є розробка математичного та програмного забезпечення пошуку клонів коду з використанням методів семантичного аналізу програм.

III. Метод пошуку клонів на основі семантичного аналізу

Запропонований підхід заснований на семантичному аналізі програми. Пошук максимально схожих підграфів проводиться в чотири етапи, що дозволяє ефективно і точно вирішувати задачу. Спочатку граф залежностей програми поділяються на підграфи (одиниці порівняння - ОП), що розглядаються як потенційні клони один іншого. Обробка всіх пар ОП проводиться в дві фази. На першій фазі за лінійний час відсіваються завідомо несхожі пари ОП. Якщо пара ОП не була відсіяна, до неї на другій фазі застосовується наближений алгоритм пошуку максимально схожих підграфів. Після того як максимально схожі підграфи знайдені, проводиться фільтрація помилкових спрацьовувань шляхом перевірки рядків вихідного коду, відповідних схожим підграфам. Якщо рядки фрагмента вихідного коду, відповідні знайденому підграфу, знаходяться на відстані, меншій p (параметр, заданий користувачем) і довжина фрагмента більша за розмір мінімального клону (задається користувачем), то вони вважаються клонами.

На рисунку 1 ілюструється робота алгоритму після того, як вершини були відсортовані за відповідними рядками вихідного коду.

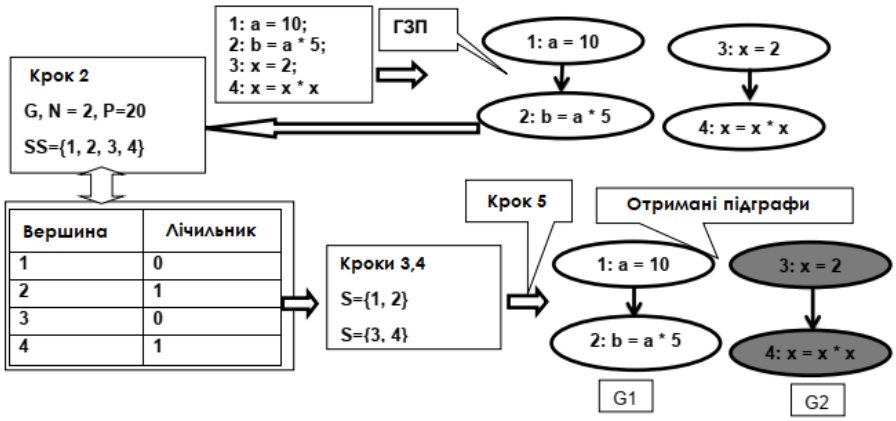

Рисунок 1 - Приклад роботи алгоритму розподілу графу залежності програми на одиниці порівняння

Складність запропонованого алгоритму $O(mn^2)$, де n – кількість вершин графа, m – середній степінь вершин.

IV. Експериментальні дослідження

Для тестування було обрано такі проекти звідкритими початковими кодами: Linux, Firefox Mozilla, LLVM/Clang iOpenSSL. Linux є операційною системою сімейства Unix. Firefox Mozillaє браузерним движком, розробленим компанією Mozilla. LLVM/Clang - компіляторна інфраструктура. В даний момент вона широко використовується як для оптимізації, так і для різних аналізів програм. OpenSSL є криптографічного бібліотекою, яка підтримує безпеку на рівні транспортування і сокетів (Transport Layer Security - TLS і Secure Sockets Layer - SSL).

На рисунку 2 наводиться порівняння знайдених клонів коду в залежності від алгоритму. Завдяки застосуванню розробленого алгоритму, кількість знайдених клонів коду для ядра Linux зросла з 913 (алгоритм IST) до 1965. Для проектів Firefox Mozilla, LLVM / Clang і OpenSSL кількість знайдених клонів зросла відповідно з 485, 62, 19 до 708, 134, 50. При застосуванні алгоритму WCC кількість знайдених клонів коду для Linux, Firefox Mozilla, LLVM / Clang становить 628, 351, 31 і 17 відповідно.

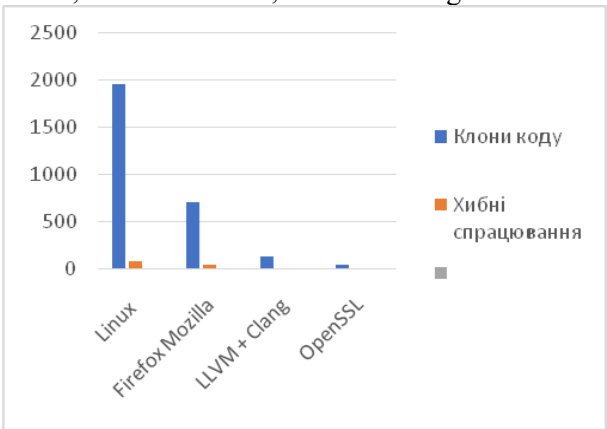

Рисунок 2–Порівняння алгоритмів

На рисунку 3 показано час роботи алгоритму. Розмір мінімального клону - 25 рядків, схожість клонів - більше 90 відсотків. Ядро Linux аналізується за 34.43 години. Час аналізу Firefox Mozilla, LLVM / Clang і OpenSSL становить 15.81, 3.52 та 0.023 години відповідно.

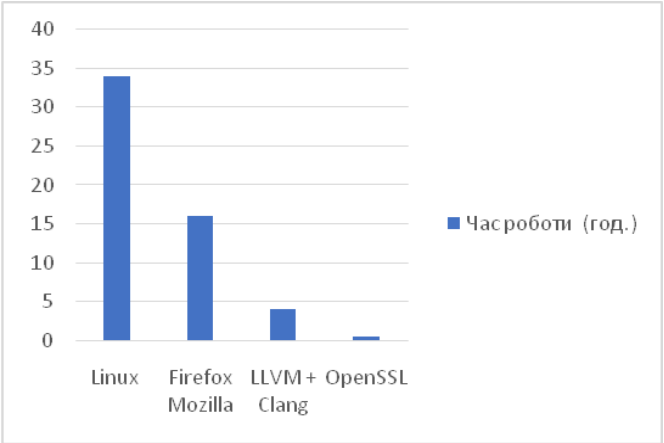

Рисунок 3 – Час роботи алгоритмів

Висновок

Проведений аналіз існуючих методів пошуку клонів коду і пошуку семантичних помилок, що виникають при неправильному копіюванні вихідного коду. Запропоновано метод побудови набору графу залежностей коду проекту на основі проміжного представлення компіляторної інфраструктури LLVM, що дозволяє отримувати набір ГЗП під час компіляції проекту без додаткових витрат.

Список використаних джерел

1. Li Z.O., Sun J. A metric space based software clone detection approach // 2nd International Conference on Software Engineering and Data Mining. 2010. С.111-116.
2. Basit H., Jarzabek S. A data mining approach for detecting higher-level clones in software // IEEE Transactions on Software Engineering. 2009. Т. 35. № 4. С.497-514.
3. Yoshiki Higo, Shinji Kusumoto MPAnalyzer: a tool for finding unintended inconsistencies in program source code // 29th ACM/IEEE international conference on Automated software engineering. 2014. С. 843-846