АЛГОРИТМИ КЛАСТЕРИЗАЦІЇ ТА МОДУЛЯРИЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

УДК 621.382

БАЗИЛЕВИЧ Роман Петрович
професор кафедри програмного забезпечення Національного університету «Львівська політехніка»

БУРТНИК Роман Романович
студент Національного університету «Львівська політехніка» кафедра програмного забезпечення

ВСТУП

Розмір і складність програмних систем постійно зростає та, відповідно, збільшується і кількість програмних класів і модулів, а також кількість зв'язків між ними. Створення моделі структури системи, і збереження її актуальності після всіх змін, які виникають, є важливою задачею. Програмні системи неперервно розвиваються, вдосконалюються і доповнюються новими функціональними можливостями протягом всього часу їх експлуатації, при цьому документація чи детальний опис їх структури стають застарілими, а системні архітектори, які проектували програмну систему, недоступні для консультацій та подальшого її вдосконалення. Виникає проблема в неперервному супроводі системи, її модифікації та розвитку.

Для полегшення аналізу та супроводу системи використовують її декомпозицію на сильнозв'язні модулі чи кластери, які містять велику кількість внутрішніх зв'язків всередині та мають незначну зв'язність між собою. Під кластеризацією розуміється процес розбиття об'єктів-сутностей на групи відповідно до значень їх атрибутів. Модуляризація - це процес об'єднання сутностей у модулі, що містять сильно зв'язані між собою елементи. Таки чином, кластеризація та модуляризація програмного забезпечення – це процес об'єднання програмних сутностей (файлів з вихідним кодом, класів, компонент) у модулі [1,2].

Існує програмне забезпечення [1,3,4] для автоматичного розбиття систем на частини. Проте, воно не може здійснити кластеризації настільки якісно, як експерти, що добре розуміють її особливості. Тим не менше, існуючі програмні системи модуляризації демонструють хороші результати, полегшують подальшу її структуризацію експертом, який вже оперує певним початковим рішенням [1,4].

Кластеризація програмної системи використовується не тільки для формування статичної структури системи, а також і для динамічних структур, які використовуються програмним забезпеченням під час виконання. Наприклад, у [5] продемонстровано, як кластеризація дозволяє вирішити проблему динамічного виділення пам'яті для «збирача сміття» у мові програмування Java.

Автоматизовані системи кластеризації програмного забезпечення включають такі етапи:

1.Підготовка сутностей. Полягає у виборі і побудові сутностей, на основі яких буде проводитись модуляризація. Це, зазвичай, класи і файли з вихідних кодом. Сутностями також можуть бути бінарні коди програми та динамічна інформація, яку генерує система у часі її виконання.

2.Фільтрування або відсіювання сутностей. Існують сутності, які варто опустити, оскільки вони ускладнюють процес модуляризації, а також, привносять зв'язки, якими можна знехтувати. Це можуть бути різного роду утиліти, або набір базових файлів, які включаються всіма іншими сутностями. Тобто, вони мають мінімальну кількість залежностей з іншими сутностями, хоча майже всі сутності залежать від них.

3.Побудова графу зв'язностей. Результатом цього етапу є орієнтований або неорієнтованих граф залежностей між сутностями. Цими залежностями можуть бути

виклики функцій іншого модуля, включення файлів, використання змінних, макросів.

4.Кластеризація сутностей. За допомогою спеціалізованих алгоритмів граф зв'язностей, побудований на попередньому етапі, розбивається на частини (кластери), які можуть мати як ієрархічну, так і інші структури [2].

5.Візуалізація кластерів. Результат попереднього етапу відображається у зручній для сприйняття людиною формі. Це дає змогу експерту внести поправки у структуру системи, виділити кожен модуль, здійснити декомпозицію системи [2,4].

6.Опрацювання результатів користувачем. Інформація, якою володіє користувач систем, є важливою для подальшого вдосконалення структури системи. Додаткові потреби, обмеження та побажання проектантів можна проаналізувати і використати для створення нових чи модифікації існуючих алгоритмів модуляризації [2,3,4].

ФОРМУЛЮВАННЯ ЗАДАЧІ

Опишемо існуючі алгоритми кластеризації сутностей. Нехай сутностями будуть файли з вихідними кодами, включення файлів одного в інший, виклик функцій, використання змінних і макросів. Для розв'язання задачі будується модель у вигляді бінарного графу, вершини якого відповідають сутностям, а ребра - залежностям між ними. Результатом є орієнтований **Граф Залежностей Модулів (ГЗМ)** [3,4]. Приклад графа зображений на рис. 1[4]. Кластеризацію цього графу можна звести до класичної комбінаторної задачі його розбиття на частини, яка відноситься до класу NP важких, що має факторіальну обчислювальну складність [6].

Використовуються декілька критеріїв оптимізації для розбиття графу на частини: максимізація сумарної кількості внутрішніх зв'язків, мінімізація сумарної кількості зовнішніх зв'язків, максимізація різниці цих значень та інші [6].

В [3,4] якість модуляризації (MQ - Modularization quality) оцінюється значенням, що підлягає мінімізації:

$$MQ = \begin{cases} \frac{1}{k}\sum_{i=1}^{k} A_i - \frac{2}{k(k-1)}\sum_{i,j=1}^{k} E_{i,j}, k > 1 \\ A_1, k = 1 \end{cases} \quad (1)$$

.

MQ знаходиться в межах $[-1,1]$. Для не зв'язаних модулів ця величина приймає значення -1, для кластера, у якого залежність між модулями максимальна – 1. Тут k - кількість частин, на які потрібно розбити систему, $A_i = \frac{\mu_i}{N_i^2}, E_{i,j} = \frac{\varepsilon_{i,j}}{2N_i N_j}$, де N_i - кількість модулів у підсистемі i, μ_i - кількість залежностей між модулями підсистеми i, $\varepsilon_{i,j}$ - кількість залежностей між модулями підсистеми i та j.

Рис. 1. Приклад Графу Залежностей Модулів (ГЗМ) [4]

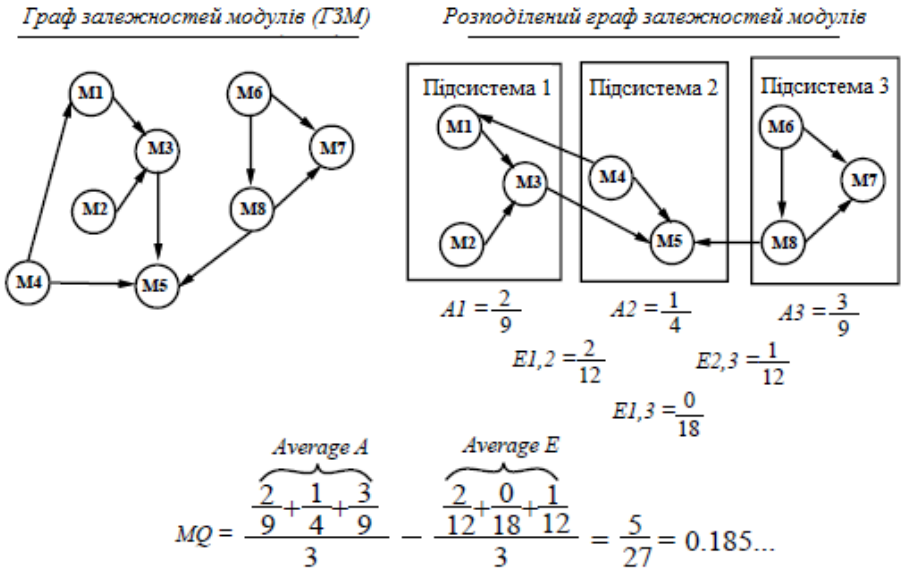

Рис. 2. Приклад визначення цільової функції MQ (Modularization quality) [4]

Алгоритми кластеризації та модуляризації. Для кластеризації та модуляризації програмного забезпечення використовують декілька алгоритмів. Найбільш поширеними є такі:

Алгоритм оптимальної кластеризації [3]. Результатом роботи алгоритму є оптимальний розв'язок. Алгоритм будує всі можливі варіанти розбиття і вибирає найкращий. Оскільки складність задачі є експоненціальною, тому на практиці алгоритм може оперувати тільки з дуже невеликою кількістю модулів. Для складних задач обчислення займає значні обсяги часу, для них необхідно використовувати наближені алгоритми.

Алгоритм сходження на вершину [2,3]. Складається з таких етапів:

1.Генерування випадкового розв'язку.

2.Переміщення елементів з одного кластера в інший для пошуку кращого розв'язку. Якщо цільова функція приймає краще значення, вибирається поточне розбиття.

3.Повторення кроку 2 до припинення покращання результату.

Алгоритм є одним із алгоритмів локальної оптимізації, його резудьтат суттєво залежить від

початкового розбиття.

Генетичний алгоритм [2,3]. Використовує типові кроки генетичних алгоритмів комбінаторної оптимізації:

1.Генерування кількох випадкових початкових розв'язків (популяції).

2.Вибір певного відсотку найкращих рішень з популяції і вдосконалити їх переміщенням елементів з одного кластера в інший.

3.Генерування нової популяції з використанням селекції і рекомбінацію модулів у кластері.

4.Повторення кроків 2,3 до припинення подальшого вдосконалень значення MQ, або доки число ітерації не перевищить задане як параметр максимальне число ітерацій.

Алгоритм показує значно кращі результати порівняно з попереднім, оскільки менше залежить від початкового розв'язку, проте вимагає значних обчислювальних затрат.

Спектральні алгоритми [2]. Будується матриця Кірхгофа, визначаються її власні значення та вектори, що використовуються для кластеризації вхідного ГЗМ. Алгоритм придатний для нескладних задач.

Алгоритм чорної діри [7]. Відноситься до алгоритмів, які створені на основі природніх процесів, використовується для ієрархічної кластеризації.

Алгоритми на основі багатоцільової оптимізації [8]. Підхід використовує дві цільових функції: для щільності і для зв'язності, які оптимізуються. Підхід дає кращі результати, ніж використання одноцільової оптимізації.

Алгоритм ієрархічної кластеризації. Для складних систем є доцільно створити ієрархію підсистем, тобто, здійснити кластеризацію кластерів. Алгоритм є впрова-

дженим в системі Bunch [1,3,4], включає такі кроки:

1.Кластеризація одним із алгоритмів для заданої початкової кількості кластерів *k*.

2.Побудова графу, в якому кластери відповідають його вершинам, а їх зовнішні зв'язки з іншими кластерами – ребрам.

3.Кластеризація новоутвореного графу.

4.Повторення кроку 2 до утворення одного кластеру.

Алгоритм має певні недоліки, оскільки в ітераціях кластеризації не використовує результати попередніх кроків, що могло б пришвидшити пошук рішення [3,4].

ЗАСТОСУВАННЯ МЕТОДУ ОПТИМАЛЬНОГО ЗГОРТАННЯ ДЛЯ КЛАСТЕРИЗАЦІЇ ТА МОДУЛЯРИЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для кластеризації програмної системи, а також її модуляризації, в тому числі ієрархічної, доцільно використовувати метод оптимального згортання схеми [6]. Підхід реалізується такими етапами:

1.Побудова дерева оптимального згортання T^R. На кожному кроці об'єднуються по два елементи за вибраним критерієм. Метою згортання схеми є виділення частин з мінімальною кількістю зв'язків між ними. Тому одним з критеріїв для об'єднання є виділення груп з максимальною кількістю зв'язків, які входять у виділену групу.

2.Аналіз дерева згортання з вибором такого його перетину, який відповідає заданим обмеженням. Множина утворених кластерів розглядаються як початковий розв'язок. Такими обмеженнями можуть виступати: кількість елементів у кожній частині, число частин, наявність у частинах певних елементів, або їх відсутність та інші. Утворений перетин - це початкова декомпозиція схеми.

3.Оптимізація розв'язку. Для покращання характеристик декомпозиції, зокрема зменшення кількості зв'язків між частинами, запропоновано декілька методів оптимізації [9,10]. Метод рекурсивних обмінів ієрархічно вбудованих кластерів [9] реалізує переміщення між довільним кластерами різних частин початкового розбиття. Метод оптимізації шляхом ітеративного перепризначення граничних елементів [10] реалізує обмін між елементами та кластерами різних частин, які мають спільні зв'язки у двох чи більшій кількості частинах, тобто є суміжними. Обидва підходи мають свої особливості, їх доцільно використовувати сумісно, тобто послідовно, що дозволить підвищити якість розв'язків.

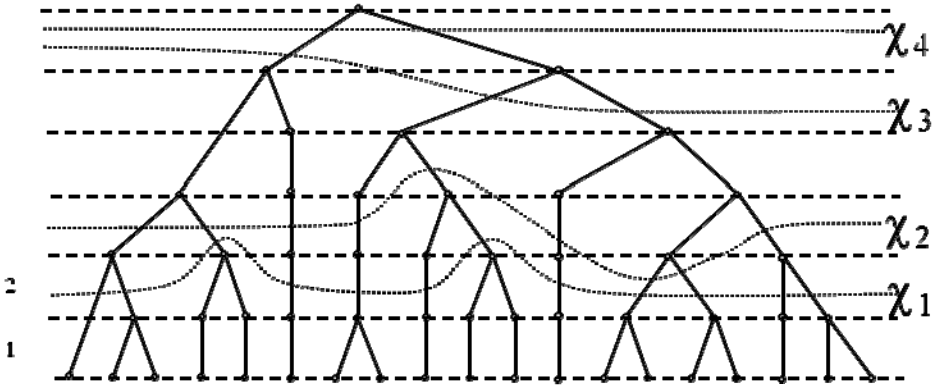

Рис. 3. Дерево оптимального згортання T^R.

Довільний перетин дерева згортання будь-якою лінією формує деяку декомпозицію системи, яку можна розглядати як початкову кластеризацію чи модуляризацію. Так, перетин χ_1 утворює частини, у які не входить більше двох елементів, перетин χ_2 утворює групи з числом елементів не більше трьох. Перетини χ_3 та χ_4 виділяють групи елементів, що розділять всю систему на три та дві «натуральні» частини відповідно. Аналіз дерева оптимального згортання T^R створює умови для отримання різних варіантів декомпозиції системи, та вибору тих, що відповідають бажаним обмеженням, в тому числі вимогам ієрархічної структури. Після отримання початкового розв'язку (кластеризації, модуляризації, в тому числі ієрархічної) доцільно здійснити його оптимізацію методами, запропонованими в [9,10], що покращить його якість.

ОСНОВНІ РЕЗУЛЬТАТИ ТА ВИСНОВКИ

Кластеризація та модуляризація програмного забезпечення є важливою і актуальною задачею для проектування, супроводу та вдосконалення існуючих програмних систем. Вона потрібна також для покращення аналізу системи та полегшення її модифікації. Задача зводиться до класичної декомпозиції бінарного графу, яка за своєю складністю відноситься до важковирішуваних комбінаторних задач класу NP, де точний розв'язок можна отримати тільки у випадку їх невеликих розмірностей. Тому більшість алгоритмів є наближеними, якість їх залежить від використаної методології генерування розв'язків. Для отримання якісних розв'язків пропонується застосувати метод оптимального згортання схеми, який підтвердив свою високу ефективність для ряду задач декомпозиції, що описуються графовими структурами. Для підвищення якості доцільно також використовувати додатково спеціалізовані оптимізаційні алгоритми. Подальші роботи будуть спрямовані на розроблення програмної системи для структурного аналізу, кластеризації та модуляризації ПЗ на основі методу оптимального згортання схеми.

ЛІТЕРАТУРА:

1. Harman, Mark. The current state and future of search based software engineering // 2007 Future of Software Engineering. IEEE Computer Society, 2007. – pp. 342 - 357.
2. Shtern, Mark, and Vassilios Tzerpos. Clustering methodologies for software engineering // Advances in Software Engineering 2012. – pp. 1-18.
3. Mancoridis, Spiros, et al. Using Automatic Clustering to Produce High-Level System Organizations of Source Code // IWP, 1998. - pp. 45-52.
4. S. Mancoridis, B. Mitchell, Y. Chen, and E. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures // Proceedings of the 15th International Conference on Software Maintenance. - Oxford, England, 1999. - pp. 50–59.
5. M. Cohen, S. B. Kooi, and W. Srisa-an. Clustering the heap in multi-threaded applications for improved garbage collection // GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, vol. 2, Seattle, Washington, 8-12 July 2006, ACM Press. - pp. 1901–1908.
6. Bazilevich R.P. Dekompozicionnye i topologicheskie metody avtomatizirovannogo kostruirovanija jelektronnyh ustrojstv.- L'vov: Vishha shkola, 1981.-168 s.
7. Jeet, Kawal, and Renu Dhir. Software Architecture Recovery using Genetic Black Hole Algorithm // ACM SIGSOFT Software Engineering Notes 40.1, 2015. - pp. 1-5.
8. Kata Praditwong, Mark Harman, Xin Yao. Software Module Clustering as a Multi-Objective Search Problem // IEEE Transactions on Software Engineering, 2011, - Vol. 37, № 2. - pp. 264-282.
9. R. Bazylevych, I. Podolskyy and L.Bazylevych. Partitioning optimization by recursive moves of hierarchically built clusters // Proc. of 2007 IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems. April, 2007, Krakow, Poland. - pp. 235 –238.
10. Roman Bazylevych, Dmytro Yanush. Partitioning optimization by iterative reassignment of the hierarchically built clusters with border elements // 2nd Mediterian Conference on Embedded Computing, MESO 2013, 15-20 June, 2013, Budva. - pp. 219 - 221.
11. Obshhaja dolja zaimstvovanij:8% (168/2196) 003% (071/2196): http://dissertation.com.ua/node/676518