УДК 004.89

В. В. Любченко, д-р техн. наук,
Є. В. Берлізов

АЛГОРИТМ ЗНАХОДЖЕННЯ ПАРЕТО-ОПТИМАЛЬНОГО РІШЕННЯ ЗАДАЧІ НАСТУПНОГО РЕЛІЗУ

Анотація. *Розглянуто три варіанти алгоритму побудови Парето-фронту для застосування їх з метою вирішення задачі наступного релізу на основі моделі двокритеріальної оптимізації. Показано, що алгоритм повного перебору з фільтруванням є кращим алгоритмом за критеріями часу розрахунків і витрат пам'яті.*

Ключові слова: *ітеративна методологія, розробка програмного забезпечення, задача наступного релізу, Парето-оптимальне рішення, двокритеріальна оптимізація, Парето-фронт, алгоритм перебору комбінацій*

V. Liubchenko, ScD.,
Y. Berlizov

THE ALGORITHM OF FINDING PARETO-OPTIMAL SOLUTIONS FOR NEXT RELEASE PROBLEM

Abstract. *We consider three versions of the algorithm of Pareto-front building for their application to the next release problem based on the model of bi-objective optimization. It is shown that the algorithm of combinations enumeration with filtering is the best algorithm accordingly of criteria of computation time and memory consumption.*

Keywords: *Iterative Methodology, Software Development, Next Release Problem, Pareto Optimal Solution, Bi-objective Optimization, Pareto-front, Algorithm of Combinations Enumeration*

В. В. Любченко, д-р техн. наук,
Е. В. Берлизов

АЛГОРИТМ ПОИСКА ПАРЕТО-ОПТИМАЛЬНОГО РЕШЕНИЯ ЗАДАЧИ СЛЕДУЮЩЕГО РЕЛИЗА

Аннотация. *Рассмотрены три варианта алгоритма построения Парето-фронта для применения их с целью решения задачи следующего релиза на основе модели двухкритериальной оптимизации. Показано, что алгоритм полного перебора с фильтрацией является лучшим алгоритмом по критериям времени расчетов и расходов памяти.*

Ключевые слова: *итеративная методология, разработка программного обеспечения, задача следующего релиза, Парето-оптимальное решение, двухкритериальная оптимизация, Парето-фронт, алгоритм перебора сочетаний*

Вступ. Останнім часом в розробці програмного забезпечення домінуючу роль відіграють ітеративні та інкрементальні методології, зокрема, методології Agile [1]. При цьому володар продукту (Product Owner) часто зустрічається з задачею вибору функціональних вимог на наступну ітерацію розробки, яку називають задачею наступного релізу (ЗНР). Він має прийняти рішення, яке буде максимально задовольняти потреби зацікавлених осіб (Stakeholders) і при цьому не буде занадто складним для реалізації впродовж ітерації [2].

Для ЗНР неможливо знайти рішення, яке буде оптимально за всіма критеріями, оскільки ці критерії не узгоджуються один з одним. Неможливо виконати всі вимоги за одну ітерацію через складність розробки, а виконавши тільки частину вимог не вдасться задовольнити потреби всіх зацікавлених осіб. Потрібно знайти компромісне за різними критеріями рішення, яке є найкращим з можливих варіантів, тобто Парето-оптимальне рішення [3].

Знаходження Парето-оптимального рішення є NP-повною задачею [4]. В роботах [5 – 7] розглянуті різні аспекти застосування емпіричних засобів для рішення ЗНР. Всі вони призводять до знаходження квазі-оптимального рішення, якість якого залежить від якості множини потенційних рішень.

Отже виникає потреба в алгоритмі генерації всіх комбінацій вимог, який є прийнятним з точки зору часу розрахунків та потрібних обчислювальних ресурсів.

Метою роботи є визначення алгоритму повного перебору комбінацій вимог для знаходження Парето-оптимального рішення ЗНР, що дозволяє скоротити витрати часу і потрібних обчислювальних ресурсів.

Задача наступного релізу як задача двокритеріальної оптимізації. Для рішення ЗНР потрібно описати функціональні вимоги на розробку так, щоб віддзеркалити точки зору зацікавлених осіб та розробників [8]. Введемо до розгляду два показники: важливість та складність. Використання цих показників в ЗНР використовується часто та було вперше запропоновано в роботі [5].

Важливість відповідає рівню задоволення інтересів зацікавленої особи у разі реалізації відповідної вимоги. Для обчислення важливості вимоги необхідно, щоб кожна з зацікавлених осіб, $C=\{c_1, \dots, c_m\}$, визначила важливість кожної з вимог, $T=\{t_1, \dots, t_n\}$. Позначимо цю оцінку $value(t_i, c_j)$. Для спрощення виконання оцінювання кожна зацікавлена особа визначає важливість вимог за 10-бальною шкалою, де 0 відповідає непотрібній вимогі, 9 – критично важливій вимогі. Інтереси зацікавлених осіб мають різне значення для компанії-розробника, що віддзеркалюється за допомогою вагових коефіцієнтів, $Weight = \{w_1, \dots, w_m\}$,

де $w_i \in [0,1]$ та $\sum_{j=1}^{m} w_j = 1$. Тоді важливість окремої вимоги розраховується як

$$imp_i = \sum_{j=1}^{m} w_j \cdot value(t_i, c_j).$$

Оцінювання складності реалізації вимог, $SP = \{sp_1, ..., sp_n\}$, виконується розробниками також за 10-бальною шкалою, де 0 відповідає вимогі, для якої можливе повторне використання існуючого рішення, 9 – критично складній для реалізації вимогі.

Вектор $\vec{x} = (x_1,...,x_n)$, де $x_i \in \{0,1\}, i = \overline{1,n}$, визначатиме, які з вимог будуть обрані в розробку на наступній ітерації. Елемент x_i дорівнює 1, якщо вимога i включена до розробки на наступній ітерації, і дорівнює 0 в іншому випадку.

Компанія-розробник зацікавлена в максимізації загальної важливості вимог і мінімізації їх загальної складності на кожній ітерації. Відомо [3], що Парето-фронт визначається як $P(Y) = \{y \in Y : \{y' \in Y : y' \succ y\} = \varnothing\}$, де відношення домінування по Парето $y' \succ y$ означає, що $y \le y', y \ne y'$. Тому задача мінімізації по другому критерію має бути перетворена на задачу максимізації, помножимо цільову функцію на -1. Тепер ЗНР можна записати в такому виді:

$$\begin{aligned} &\text{Maximize } f_1(\vec{x}) = \sum_{i=1}^{n} imp_i \cdot x_i \\ &\text{Maximize } f_2(\vec{x}) = -\sum_{i=1}^{n} sp_i \cdot x_i \end{aligned}.$$

Тобто ЗНР є задачею оптимізації з двома цільовими функціями на комбінаторній множині комбінацій вимог.

В цій роботі ми не враховуємо залежності між вимогами, які зазвичай існують. Але для досягнення мети роботи таке припущення не впливає на результат.

Алгоритми побудови Парето-фронту. Парето-фронт, тобто множину Парето-оптимальних рішень, можна легко знайти використовуючи алгоритм повного перебору комбінацій вимог для генерації потенційних рішень. Але це швидко призводить до проблем з потрібними для виконання розрахунків обсягами ресурсів.

Наприклад, у випадку реалізації алгоритму мовою Java, кожен об'єкт класу Task, який містить мінімальну необхідну інформацію про окрему вимогу, потребуватиме 32 байта пам'яті. Якщо як базовий клас для зберігання інформації про всі вимоги використовувати стандартну Java-колекцію ArrayList, то розмір об'єкта цього класу буде (16+4*n*), де *n* – кількість об'єктів, що зберігає колекція. За цих умов вже для генерації рішень з 26 вимог буде потрібно більше оперативної пам'яті ніж встановлено в більшості сучасних комп'ютерів [9] та значно більше ніж виділено для Java Virtual Machine [10].

В загальному випадку витрати пам'яті на зберігання елементів комбінаторної множини комбінацій вимог оцінюються як $O(2^n)$, де *n* – загальна кількість вимог на розробку. Слід також враховувати витрати пам'яті на побудову Парето-фронту. З цією метою застосовується алгоритм NSGA–II [11], оцінка витрат пам'яті якого складає $O(q \log q)$, де *q* – кількість комбінацій вимог, на яких будується Парето-фронт.

Розглянемо три варіанти алгоритму побудови Парето-фронту на основі повного перебору комбінацій вимог.

Алгоритм 1 (повний перебір комбінацій з попередньою генерацією).

1. Для всіх $k \le n$ згенерувати всі можливі комбінації *k* вимог з *n* можливих за допомогою рекурсивного алгоритму генерації комбінацій [12].

2. Запустити алгоритм NSGA–II на всіх згенерованих комбінаціях.

Результати роботи алгорітму показані на рис. 1 (показники важливості і складності виміряні в балах).

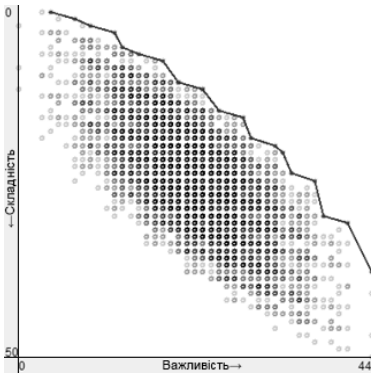

Рис. 1. Приклад побудованого за допомогою алгоритму попередньої генерації Парето-фронту на 13 вимогах

Обчислювальна складність алгоритму становить $O(2^n) + O(2^n \log 2^n)$. Усі згенеровані для побудови Парето-фронту комбінації одночасно зберігаються в пам'яті комп'ютера, проте їх можна використовувати для інших розрахунків чи аналізу. Алгоритм досить швидкий, оскільки немає потреби в частому виклику збирача сміття, а також через швидкий доступ до будь-якого елементу масиву.

Алгоритм 2 (повний перебір комбінацій з генерацією в реальному часі).

1. Задати $k=1$.

2. Згенерувати всі можливі комбінації *k* вимог з *n* можливих за допомогою рекурсивного алгоритму генерації комбінацій.

3. Запустити алгоритм NSGA–II на згенерованих комбінаціях та раніше побудованому Парето-фронті.

4. Перевірити виконання нерівності $k \le n$. Якщо вона виконується, то призначити $k=k+1$ та повернутися до кроку 2. Інакше завершити розрахунки.

Обчислювальна складність алгоритму така ж, як і у попереднього. Реалізація алгоритму мовою Java дозволяє скоротити потрібний для програми об'єм оперативної пам'яті, але потребує активної роботи збирача сміття. Тому економія пам'яті, яка дозволить збільшити граничну кількість вимог на 2-3 одиниці, обійдеться суттєвим

збільшенням часових витрат. Це пов'язано з різними чинниками, одним з яких є відсутність динамічної компіляції. Реалізація алгоритму низькорівневими мовами програмування покращить ситуацію, але несуттєво.

Алгоритм 3 (повний перебір комбінацій з попередньою генерацією та фільтруванням). Скористаємося тим, що ми знаємо про ЗНР. Парето-фронт будується за критеріями важливості та складності, тому максимальній розмір фронту буде дорівнювати меншому з розмірів критеріїв.

1. Створити масив для зберігання комбінацій вимог довжиною, що дорівнює складності комбінації зі всіх вимог.

2. Задати $k = 1$.

3. Запустити алгоритм генерації комбінації k вимог з n можливих.

4. Коли алгоритм генерує нову комбінацію, обчислити відповідну їй двійку оцінок $(f_1(\vec{x}), f_2(\vec{x}))$.

5. Додати комбінацію у комірку масиву, індекс якої відповідає її складності. Якщо комірка не пуста, то порівняти попередню і нову комбінацію за важливістю та залишити ту з них, важливість якої більше.

6. Якщо всі комбінацій з довжиною k згенеровано, то призначити $k=k+1$ та перейти до кроку 7, інакше повернутися до кроку 4.

7. Якщо $k>n$, то перейти до кроку 8, інакше повернутися до кроку 3.

8. Запустити алгоритм NSGA–II на масиві згенерованих комбінацій.

Цей алгоритм відрізняться від алгоритму 1 тим, що згенеровані комбінації проходять фільтрацію, залишаються лише ті з них, які найбільш близькі до Парето-фронту.

Обчислювальна складність алгоритму становить $O(2^n)+O(q)$, де q – складність комбінації зі всіх вимог. Алгоритм має низькі витрати ресурсів пам'яті та високу швидкість виконання розрахунків. Парето-фронт розраховується на меншій кількості комбінацій, ніж у попередніх алгоритмах, що істотно зменшує час його побудови.

На рис. 2 наведено приклад Парето-фронту, побудованого алгоритмом: точки відповідають комбінаціям, які аналізуються, а лінія – побудованому Парето-фронту. Показники важливості і складності, як і раніше, виміряні в балах.

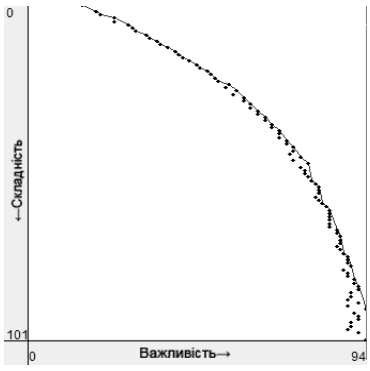

Рис. 2. Приклад побудованого за допомогою алгоритму попередньої генерації с фільтруванням Парето-фронту на 20 вимогах

Порівняємо розглянуті алгоритми між собою і визначимо найбільш придатний для вирішення ЗНР.

На рис. 3 показані графіки часу потрібного на виконання розрахунків різними алгоритмами, отримані як результат експериментів.

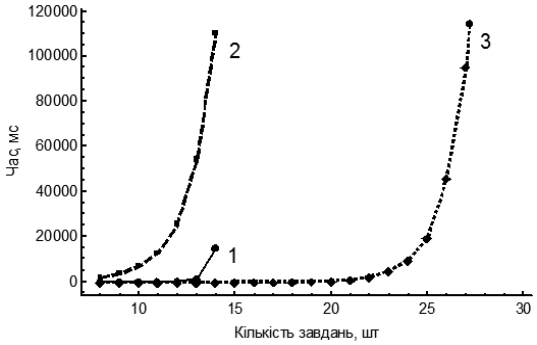

Рис. 3. Графіки часу розрахунків за (1) – за алгоритмом 1; (2) – за алгоритмом 2; (3) – за алгоритмом 3

Як видно з рис. 3, алгоритм 3 демонструє значно кращі результати за швидкістю виконання обчислень, ніж алгоритми 1 і 2.

На рис. 4 показані графіки споживання ресурсів пам'яті різними алгоритмами при різних розмірах ЗНР, отримані як результат експериментів.

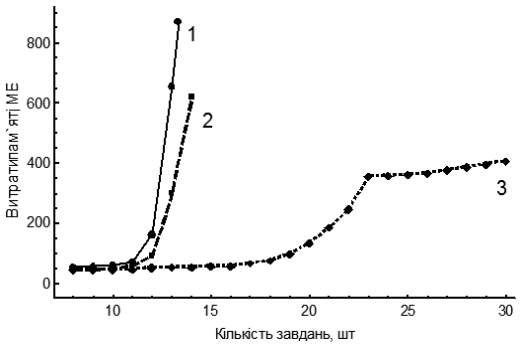

Рис. 4. Графіки споживання ресурсів пам'яті (1) – за алгоритмом 1; (2) – за алгоритмом 2; (3) – за алгоритмом 3

Як видно, споживання ресурсів алгоритмами 1 і 2 швидко зростає, і вже для 15 вимог стає неприйнятним. Тоді як алгоритм 3 має цілком прийнятні потреби в пам'яті. Злам відповідного графіка можна пояснити активною роботою збирача сміття, який викликається при нестачі памяті.

Висновки. В роботі розглянуті три варіанти алгоритму побудови Парето-фронту для рішення ЗНР. З порівняння властивостей аналізованих варіантів алгоритмів (рисунки 3 та 4) можна зробити висновок, що кращим алгоритмом за часом розрахунків і витратами пам'яті є алгоритм повного перебору комбінацій з попередньою генерацією та фільтруванням.

Список використаної літератури

1. West D., and Grant T., (2010), Agile Development: Mainstream Adoption Has Changed Agility Cambridge: Forrester Research, 22 p.

2. Bagnall A.J., Rayward-Smith V. J., and Whittley I. M., (2001), The Next Release Problem, *Information and Software Technology*, Vol. 43, pp. 883 – 890.

3. Ногин В. Д. Принятие решений в многокритериальной среде: количественный подход [Текст] / В. Д. Ногин. – М. : ФИЗМАТЛИТ, 2002. – 176 с.

4. Колечкина Л. Н. Оптимальные решения многокритериальных комбинаторных задач на размещениях / Л. Н. Колечкина // Теорія оптимальних рішень. – К. : – 2007. – № 6. – С. 67 – 73.

5. Zhang Y., Harman M., and Mansouri S.A., (2007), The Multi-objective Next Release Problem, *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07),* London, UK, pp. 1129 – 1137.

6. Cai X., Wei O., and Huang Z., (2012), Evolutionary Approaches for Multi-objective Next Release Problem, *Computing and Informatics*, Vol. 31, pp. 847 – 875.

7. Durillo J.J., Zhang Y., Alba E., Harman M., and Nebro A.J., (2010), A Study of the Bi-objective Next Release Problem, *Empirical Software Engineering*, pp. 1 – 32, Doi: 10.1007/s10664-010-9147-3.

8. Greer D., and Ruhe G., (2004), Software Release Planning: an Evolutionary and Iterative Approach, *Information and Software Technology*, Vol. 46, pp. 243 –253.

9. Howard M., (2011), Growth of DRAM Content in Notebook PCs Decelerates Partly Due to Rise of Ultra books and the Cloud, IHS Technology, Url: https://technology.ihs.com/389419/growth-of-dram-content-in-notebook-pcs-decelerates-partly-due-to-rise-of-ultrabooks-and-the-cloud (Дата доступу: 08.05.2015)

10. Java SE HotSpot Virtual Machine Garbage Collection Tuning Guide, (2015), Oracle Corporation, Url: http://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/ (Дата доступу: 08.05.2015)

11. Deb K., Pratap A., Agarwal S., and Meyarivan T., (2002), A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II *IEEE Transactions on Evolutionary Computation*, Vol. 6(2), pp. 182 – 197.

12. Ruskey F., (2003), Combinatorial Generation CSC-425/520 Univ. of Victoria CA, 39 p., Url: http://www.1stworks.com/ref/CombGen.pdf (Дата доступу: 08.05.2015)

Отримано 28.05.2015

References

1. West D., and Grant T., (2010), Agile Development: Mainstream Adoption Has Changed Agility Cambridge: Forrester Research, 22 p.

2. Bagnall A.J., Rayward-Smith V.J., and Whittley I.M., (2001), The Next Release Problem, *Information and Software Technology*, Vol. 43, pp. 883 –890.

3. Nogin V.D. Priniatie reshenij v mnogokriterial'noj srede: kolichestvennyi podkhod [Decision-making in Multi-criteria Environment: Quantitative Approach], (2002), Moscow, Russian Federation, 176 p. (In Russian).

4. Kolechkina L.N. Optimal'nye reshenia mnogokriterial'nykh kombinatornykh zadach na razmescheniakh [Optimum Decisions of Multicriterion Combinatorial Problems on Placing], (2007), *Teoria Optymal'nykh Rishen'*, Kiev, Ukraine, Vol. 6, pp. 67 – 73 (In Russian).

5. Zhang Y., Harman M., and Mansouri S.A., (2007), The Multi-objective Next Release Problem, *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07),* London, UK, pp. 1129 – 1137.

6. Cai X., Wei O., and Huang Z., (2012), Evolutionary Approaches for Multi-objective Next Release Problem, *Computing and Informatics*, Vol. 31, pp. 847 – 875.

7. Durillo J.J., Zhang Y., Alba E., Harman M., and Nebro A.J., (2010), A Study of the Bi-objective Next Release Problem, *Empirical Software Engineering*, pp. 1 – 32, Doi: 10.1007/s10664-010-9147-3.

8. Greer D., and Ruhe G., (2004), Software Release Planning: an Evolutionary and Iterative Approach *Information and Software Technology*, Vol. 46, pp. 243 – 253.

9. Howard M., (2011), Growth of DRAM Content in Notebook PCs Decelerates Partly Due to Rise of Ultra books and the Cloud, IHS Technology, Url: https://technology.ihs.com/389419/growth-of-dram-content-in-notebook-pcs-decelerates-partly-due-to-rise-of-ultrabooks-and-the-cloud (accessed 08.05.2015).

10. Java SE HotSpot Virtual Machine Garbage Collection Tuning Guide, (2015), Oracle Corporation, Url: http://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/ (accessed 08.05.2015).

11. Deb K., Pratap A., Agarwal S., and Meyarivan T., (2002), A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II *IEEE Transactions on Evolutionary Computation*, Vol. 6(2), pp. 182 – 197.

12. Ruskey F., (2003), Combinatorial Generation CSC-425/520 Univ. of Victoria CA, 39 p., Url: http://www.1stworks.com/ref/CombGen.pdf (accessed 08.05.2015).

Любченко
Віра Вікторівна,
д-р. техн. наук, проф. каф. системного програмного забезпечення Одеського нац. політехнічного ун-ту,
тел. (048) 705 8675.
E-mail: lvv@edu.opu.ua

Берлізов
Євгеній Володимирович,
студент Одеського нац. політехнічного ун-ту,
тел. (091) 988 21 44.
E-mail: berlizov@me.com