УДК 004.25, 004.383

І. І. Лопіт
Національний університет "Львівська політехніка",
кафедра безпеки інформаційних технологій

ОПТИМІЗАЦІЯ ЖАДІБНИХ АЛГОРИТМІВ ПОШУКУ ДЛЯ СКОМБІНОВАНИХ ПОСЛІДОВНОСТЕЙ ДАНИХ

Розглянуто питання оптимізації жадібних алгоритмів, які можуть бути застосовані для оптимізації розміщення/маршрутизації між компонентами в обчислювальних системах у випадку, коли послідовності даних було отримано за допомогою комбінаційного розподілу. Проаналізовано недоліки використання звичайного жадібного алгоритму і запропоновано його оптимізований варіант на основі упорядкованого матричного запису розміщень, який дає змогу підвищити швидкодію алгоритму у 2.7 разу для 482 унікальних елементів.

Ключові слова: алгоритми оптимізації і пошуку, жадібні алгоритми, спеціалізовані процесори.

OPTIMIZATION OF THE GREEDY SEARCH ALGORITHMS FOR COMBINED DATA SEQUENCES

The paper describes optimization of the greedy algorithm that can be used to optimize the placement / routing between components in computer systems, when the sequences of data were obtained by using combinations. The disadvantages of original greedy algorithm were analyzed and its optimized version, which is based on an ordered matrix notation to store permutation, was proposed. This approach increases algorithm performance in 2.7 times for 482 unique items.

Keywords: search and optimization algorithms, greedy algorithms, application-specific processors.

Вступ

Тенденції до зниження темпів розвитку елементної бази комп'ютерних систем вперше з'явилися на початку нового тисячоліття. У своїй статті *No Exponential is Forever: But "Forever Can Be Delayed"* [1] Гордон Мур зазначив, що експоненціальний ріст не може тривати вічно, і рано чи пізно він почне уповільнюватись, а потім припиниться взагалі. Зростання обчислювальної потужності елементної бази в основному забезпечується еволюцією транзисторів та способом їх виготовлення. Проте з часом якісні перетворення транзисторів стануть неможливими або економічно невигідними внаслідок атомарної природи речей.

Сьогодні більшість виробників мікропроцесорів підвищують продуктивність комп'ютерних систем переважно екстенсивним методом, тобто збільшенням кількості ядер універсальних процесорів або їхньої частоти. За останні 5 років 4–8-ядерні процесори стали звичним явищем. Однак, не всі класи алгоритмів можуть бути розпаралелені. Також слід зауважити, що підтримка багатозадачності збільшує накладні витрати на синхронізацію між задачами, що, своєю чергою, знижує реальну продуктивність системи. Збільшення тактової частоти теж не завжди є дієвим засобом. Залежність між тактовою частотою, тепловиділенням та енергоспоживанням є експоненціальною, і для того, щоб універсальний процесор міг працювати на частотах, вищих за 6 ГГц, необхідні складні технологічні процеси для підтримання сталої низької температури.

Одним з можливих рішень є відмова від екстенсивного шляху розвитку обчислювальних систем. Інтенсивний шлях, натомість, передбачає використання спеціалізації та оптимізації компонентів комп'ютерних систем.

Алгоритми оптимізації та пошуку в більшості випадків являють собою послідовність кроків, на кожному з яких надається деякий простір можливих виборів [2]. Сьогодні можна виокремити два основні принципи, які можна використовувати для розв'язання задач оптимізації: принцип динамічного програмування та принцип жадібних алгоритмів.

При використанні динамічного програмування необхідно робити вибір на кожному етапі, який зазвичай залежить від розв'язання підзадач. Методом динамічного програмування задачі зазвичай розв'язують у висхідному напрямі, тобто спочатку розв'язують простіші, а потім – складніші задачі [2]. Це вносить надлишкову складність в алгоритм і є нераціональним при розв'язанні простих задач.

Своєю чергою, жадібні алгоритми завжди передбачають вибір, який видається найкращим на цьому етапі, за припущення, що послідовність виконання найкращих виборів на кожному етапі приведе до оптимального розв'язку глобальної задачі [2]. Жадібні алгоритми не завжди призводять до оптимального розв'язку задачі оптимізації, проте вони є простими та ефективними для широкого класу задач: алгоритми пошуку мінімальних остових дерев (анг. ***minimum-spanning-tree***), алгоритм Дейкстри (***Dijkstra***) для пошуку найкоротшого шляху із одної точки в іншу та евристичний жадібний підхід Чватала (***Chvatal***) для задачі про покриття множин (***set-covering***).

Аналіз досліджень і публікацій

Алгоритми оптимізації, зокрема жадібні, описано в праці [2]. У цій праці комплексно висвітлено підходи до побудови алгоритмів оптимізації й оцінювання їх складності, які використав автор для реалізації звичайного жадібного алгоритму. В праці [3] подано загальні підходи до оцінювання характеристик алгоритмів та їх реалізації. Цю працю використав автор як базис для розроблення жадібного алгоритму оптимізації. У праці [4] висвітлено математичні основи, які було використано для побудови оптимізованого жадібного алгоритму, а саме – властивості даних, отримані комбінуванням.

Постановка проблеми

До найпоширеніших задач оптимізації у сфері комп'ютерних систем належать задачі розташування компонентів (анг. ***placing***) і маршрутизація між ними (анг. ***routing***). Для ідеального розміщення/маршрутизації компонентів необхідно перебрати всі можливі варіанти, тобто виконати повний перебір (анг. ***brute force***). Формально міжкомпонентні з'єднання і компоненти є унікальними, отже, кількість варіантів для повного перебору дорівнює ***n!***. Факторіальна складність повного перебору унеможливлює його застосування на практиці.

Цю проблему вирішують, використовуючи підхід "розділяй і володарюй" (анг. ***divide and conquer***) [3]. Наприклад, задачу повного перебору розміщення компонентів можна представити як серію часткових переборів – шукаючи поєднання груп елементів, яке є оптимальним, а на кожному наступному кроці використовувати оптимальні поєднання як основу.

Так чи інакше, генерування можливих комбінацій розташувань/маршрутів і пошук оптимальних значень часто зустрічається в задачах оптимізації комп'ютерних компонентів. Проте використання жадібного алгоритму у цьому випадку має свою особливість: перед вибором наступного розміщення / міжкомпонентного з'єднання необхідно переконатись, що воно не містить і компонентів, які до цього були використані. Цього можна досягти декількома шляхами, одним з яких є додатковий крок для маркування груп, що містять вже використані компоненти. Це, своєю чергою, призводить до зростання складності алгоритму на кожному кроці. Цей недолік суттєво впливає на жадібний алгоритм, збільшуючи його складність майже вдвічі. Питання оптимізації жадібного алгоритму, який працює з комбінаціями, розглянуто у цій роботі.

1. Особливості виконання жадібних алгоритмів над комбінаціями

Генерування можливих унікальних розміщень групи елементів – це задача з області комбінаторики [3], яка має факторіальну складність. Кількість можливих комбінацій унікальних елементів можна обчислити за формулою:

$$C_n^k = \frac{n!}{k!(n-k)}, \tag{1}$$

де n – загальна кількість елементів; k – кількість елементів у розміщенні.

Наприклад, якщо кількість елементів n=100, а розмір розміщення k=2, то кількість можливих розміщень 4950. Якщо k=4, то кількість можливих розміщень 3921225. Якщо ж k=6, то кількість можливих розміщень 1192052400. Зважаючи на ці числа, стає зрозумілим, що використання k>2 суттєво збільшує кількість комбінацій, тому доцільно використовувати значення k=2.

Жадібні алгоритми, які працюють над комбінаціями, мають дві основні характеристики: кількість елементів n і кількість спроб пошуку S. Кількість спроб S можна обчислити за формулою:

$$S = n/k. \tag{2}$$

Якщо кількість спроб менша за це число, то не всі елементи буде обрано, а в протилежному випадку – деякі елементи буде обрано за два і більше разів, а це суперечить принципу оптимізації, позаяк вносить надлишковість.

2. Складність жадібних алгоритмів над комбінаціями

Однією з найголовніших характеристик будь-якого алгоритму є його складність. Розглянемо складність жадібного алгоритму пошуку у випадку, коли кількість унікальних елементів $n = R$, а розмір комбінації $k = 2$. Тоді за формулою (1) кількість комбінацій дорівнює:

$$R(R-1)/2. \tag{3}$$

Кількість необхідних для виконання жадібного алгоритму операцій можна обчислити на основі його складових так:

- пошук найменшої комбінації. Складність пошуку лінійна, в гіршому випадку необхідно перевірити всі елементи – $R(R-1)/2$ операцій;
- забезпечити унікальність вибору наступної комбінації: промаркувати комбінації, які містять використані елементи. Елементи можуть знаходитись у будь-якій частині списку, а отже, потрібно додатково виконати $R(R-1)/2$ (кількість всіх елементів) операцій, щоб встановити відповідний прапорець після того, як буде знайдено мінімальний елемент;
- необхідно знайти R/2 мінімальних груп за формулою 2.

Отже, загальну кількість операцій для жадібного пошуку за найгіршого сценарію можна визначити так:

$$\frac{R}{2}\left(\frac{R(R-1)}{2}+\frac{R(R-1)}{2}\right)=\frac{R^2(R-1)}{2}. \tag{4}$$

3. Оптимізація жадібного алгоритму пошуку

В попередньому розділі було подано вирази для обчислення складності жадібного алгоритму (формула 4). Отриманий результат можна покращити, якщо подати перестановки в упорядкованій матричній формі (УМФ). Для отримання упорядкованої матричної форми необхідно заповнити матрицю розміром R x R, так: нехай перший елемент у розташуванні означатиме індекс стовпця матриці, а другий – індекс рядка. Комбінації стовпців, отримані за формулою 2, є унікальними, а отже, ймовірність того, що у двох елементів будуть одинакові координати, дорівнює нулю.

Наприклад, сформуємо упорядковану матрицю на основі групи G для випадку, коли R дорівнює 6 (цифри в дужках біля елементів означають номери стовпців в елементі групи):

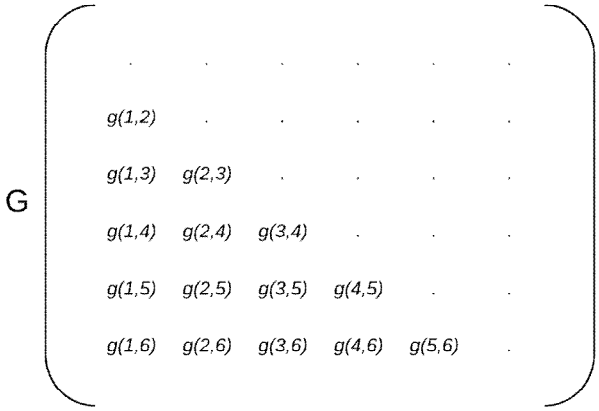

Рис. 1. Упорядкована матриця на основі групи G для випадку, коли R дорівнює 6

Ця форма має низку переваг, які дають змогу оптимізувати жадібний алгоритм пошуку.

Оптимізація встановлення доступності

Використовуючи УМФ, можна зменшити складність встановлення прапорця доступності. На рис. 2 зображено приклад швидкого обходу елементів матриці для встановлення прапорця у випадку, якщо необхідно обійти всі елементи, які містять двійку.

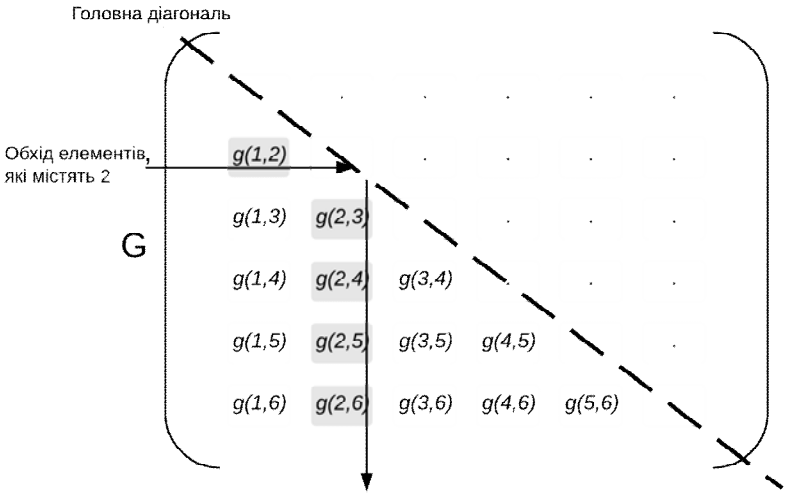

Рис. 2. Обхід упорядкованої матриці для швидкого встановлення прапорця

Алгоритм обходу наступний:

- обрати число для того, щоб позначити всі елементи, які його містять;
- обрати перший елемент зліва, індекс рядка якого дорівнює цьому числу;
- рухатись праворуч до головної діагоналі матриці;
- після досягнення головної діагоналі рухатись вертикально вниз до останнього рядка.

Кількість операцій для цього обходу дорівнює R–1, оскільки розміщення елементів відоме наперед.

Кожен елемент g групи G містить 2 стовпці, отже, щоб встановити прапорці, на кожному кроці жадібного алгоритму необхідно виконати 2 операції обходу. Загальну кількість операцій для встановлення прапорців на кожному кроці оптимізованого жадібного алгоритму визначимо за виразом:

$$2(R-1). \quad (5)$$

Оптимізація пошуку мінімального елемента

Використовуючи УМФ, можна також зменшити кількість операцій для пошуку мінімального елемента. В неоптимізованому жадібному алгоритмі на кожному кроці вона дорівнює $R(R-1)/2$ (кількість елементів), що зумовлено відсутністю впорядкованості в представленні групи *G*. Це призводить до того, що на етапі пошуку мінімального елемента відбувається надлишковий прохід елементами, для яких вже було встановлено прапорець заборони вибору. Проте, якщо використати обхід упорядкованої матриці рядок за рядком і ввести додаткову матрицю-стовпець E для контролю доступу до рядків, то кількість операцій можна зменшити (див. рис. 3).

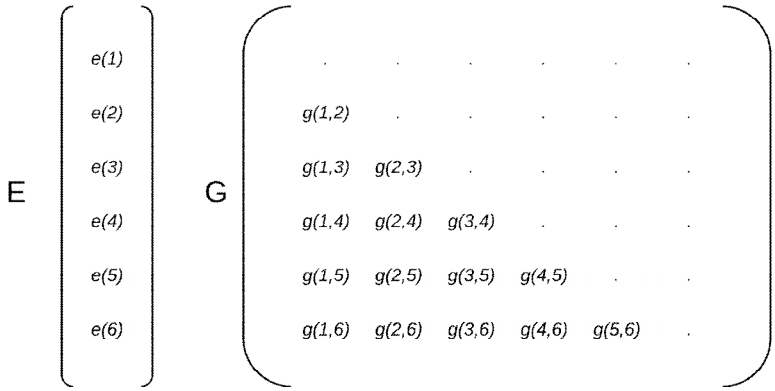

Рис. 3. Упорядкована матриця G і матриця-стовпець заборони доступу E

При переході на новий рядок спочатку зчитується дозвіл на доступ до рядка з матриці-стовпця E. У випадку, якщо прапорець встановлено, відбувається перехід на інший рядок, у протилежному випадку виконується обхід поточного. Прапорці для матриці E встановлюються разом із прапорцями для елементів матриці G.

Усі елементи в УМФ розміщені під головною діагоналлю, тобто вона є нижньою трикутною матрицею. У матриць цього типу кількість елементів у кожному рядку не є сталою, а змінюється від 0 для першого рядка до *R*–1 для *R*-го рядка з кроком 1. З використанням вказаної характеристики трикутної матриці визначимо середнє число елементів, яким елемент-прапорець *e* з матриці E забороняє доступ до елементів матриці G. Для цього використаємо формулу 9 для знаходження середнього арифметичного значення:

$$\overline{x} = \frac{1}{n} S, \tag{6}$$

де $\overline{x}$ – середнє значення; n – кількість елементів; S – сума елементів.

Кожен рядок матриці представимо як елемент арифметичної прогресії з кроком 1. Тоді загальну кількість елементів у рядках можна подати як арифметичну прогресію:

$$S = \frac{x_k + x_{k+n-1}}{2} n, \tag{7}$$

де S – сума елементів арифметичної прогресії; x_k – перший елемент арифметичної прогресії; x_{k+n-1} – останній її елемент; n – кількість елементів арифметичної прогресії.

Підставивши ці величини, визначимо середнє число елементів *D*, які забороняє елемент-прапорець *e* з матриці E:

$$D = \frac{1}{n} S = \frac{1}{n}\left(\frac{x_k + x_{k+n-1}}{2} \right) n = \frac{x_k + x_{k+n-1}}{2} = \frac{0 + R - 1}{2} = \frac{R-1}{2}. \tag{8}$$

Враховуючи те, що на кожному кроці встановлюється 2 елементи-прапорці в матриці E, то кожного кроку складність пошуку буде зменшуватись на величину

$$2\frac{R-1}{2} = R - 1. \tag{9}$$

Оптимізація останнього кроку

Під час оптимізації встановлення прапорця було встановлено, що на кожному кроці здійснюється 2 ($R-1$) операцій за формулою 9. Якщо врахувати, що кількість кроків жадібного алгоритму дорівнює R/2, то загальна кількість операцій із встановлення прапорця дорівнюватиме:

$$2(R-1)\frac{R}{2} = R(R-1). \tag{10}$$

Отже, якщо кількість елементів у матриці G дорівнює $\frac{R(R-1)}{2}$, кількість операцій для встановлення прапорця доступності завжди більша за кількість елементів. За попереднім твердженням можна припустити, що на останньому кроці можлива ситуація, коли немає необхідності робити швидкий обхід, бо відповідні прапорці вже встановлено. Це припущення можна підтвердити, якщо розглянути детальніше швидкий обхід матриці для встановлення прапорця. Існує така закономірність: кожний наступний обхід встановлює на один унікальний прапорець менше, ніж попередній (див. рис. 4).

Рис. 4 демонструє цю закономірність, адже перший обхід матриці встановив прапорець для R–1, тобто п'яти елементів, другий – для R–2, тобто чотирьох елементів, а третій – для R–3, тобто трьох елементів. Враховуючи те, що на кожному кроці необхідно виконати 2 операції обходу, а кількість кроків дорівнює R/2, необхідно R операцій обходу, а тому можемо записати закономірність у вигляді виразу:

$$U = R - N, \tag{11}$$

де U – кількість унікальних елементів, для яких буде встановлено прапорець, N – номер кроку.

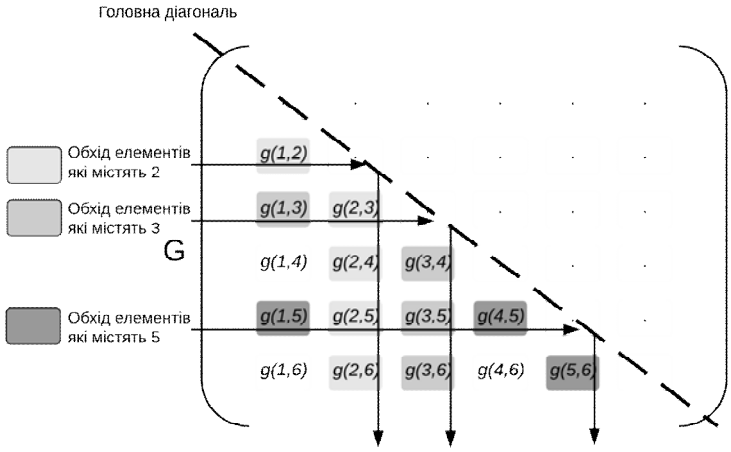

Рис. 4. Багаторазовий обхід упорядкованої матриці в наступному порядку – елементи,які містять число 2, потім 3 і 5

Якщо обчислити попередній вираз для передостаннього і останнього кроків (R–1 і R), то кількість унікальних елементів, для яких буде встановлено прапорець, дорівнюватиме 1 і 0 відповідно. Отже, ці операції є надлишковими, і на момент виконання останніх обходів матриці, тобто останнього кроку, залишатиметься один елемент, який і є мінімальним. Його координати можна виявити за матрицею E – це будуть останні два невстановлені елементи-прапорці. Складність цієї операції є такою самою, як складність лінійного пошуку елемента у матриці E, тобто кількість операцій дорівнюватиме R.

Складність оптимізованого жадібного алгоритму пошуку

Кількість операцій, необхідних для оптимізованого жадібного алгоритму пошуку, має 2 складові:

Кількість операцій для встановлення прапорців на кожному кроці за формулою (9), яка дорівнює 2(R-1), а з урахуванням додаткової операції для встановлення елементів прапорців у матриці E вона збільшиться до 2R.

Кількість операцій для пошуку мінімального елемента дорівнює $R(R\text{-}1)/2$ для першого кроку, а для кожного наступного – зменшуватиметься на R-1 (за виразом (5)). Зважаючи на цю особливість і те, що на кожному кроці додатково буде перевірятись елемент матриці E, отримаємо вираз для опису кількості операцій для пошуку мінімального елемента : $\frac{R^2}{2} - (\mathrm{R}-1)(N-1)$.

Розрахуємо кількість операцій для одного кроку оптимізованого жадібного алгоритму як суму його складових. Вона дорівнює $2R + \frac{R^2}{2} - (\mathrm{R}\text{-}1)(N-1)$.

Беручи до уваги те, що кількість операцій для пошуку мінімального елемента не є константною, а зменшується з кожним кроком, загальну кількість операцій можна обчислити як суму спадної арифметичної прогресії з визначником $d = -(R-1)$ і першим елементом $a_1 = 2(R-1) + \frac{R(R-1)}{2}$. Сума n перших елементів арифметичної прогресії визначається як:

$$S_n = \frac{2a_1 + d(n-1)}{2} n. \tag{12}$$

Врахувавши оптимізацію останнього кроку, розрахуємо кількість операцій для оптимізованого жадібного алгоритму:

$$\frac{2\times(2R + \frac{R^2}{2}) - (R-1)\times(\frac{R}{2} - 1 - 1)}{2} \times (\frac{R}{2} - 1) + R = \frac{R^3}{8} + \frac{11R^2}{8} - \frac{11R}{4} + 1 \tag{13}$$

Різницю між кількістю операцій для жадібного і оптимізованого жадібного алгоритмів пошуку наведено на рис. 5.

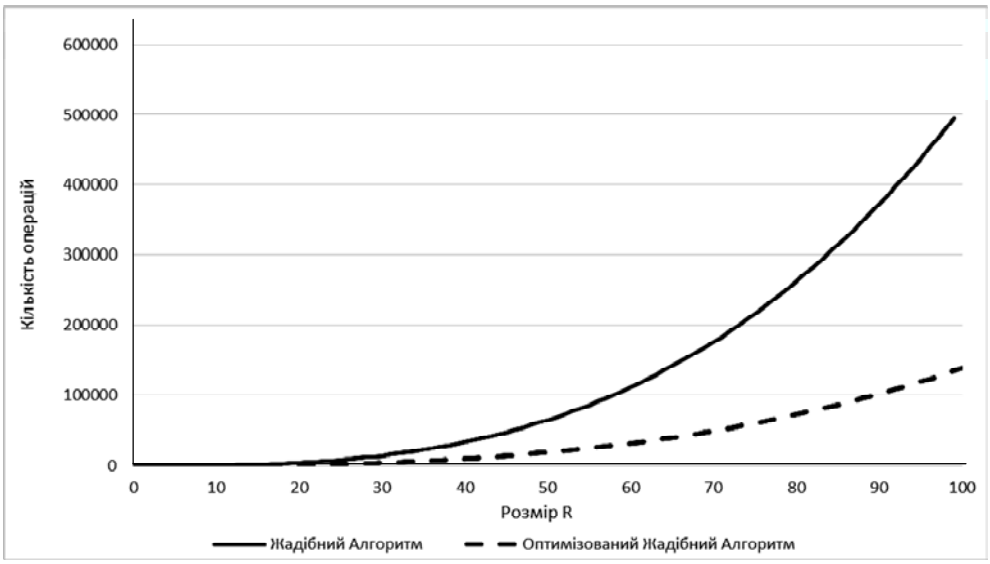

Рис. 5. Розподіл кількості операцій залежно від кількості R унікальних елементів у комбінації для жадібного і оптимізованого жадібного алгоритмів пошуку

4. Результати експериментальних досліджень

З метою підтвердження ефективності запропонованого алгоритму проведено експериментальні дослідження. Суть експерименту така: реалізацію звичайного і оптимізованого жадібного алгоритму мовою C++ виконують 20 разів, для кожної множини R значень. Множини сформовані із значень у діапазоні від 10 до 482 з кроком 4.

Як тестове обладнання використано процесор i7-4930K компанії *Intel*, як компілятор – MSVC v140 x64. Результати експериментальних досліджень наведено на рис. 6.

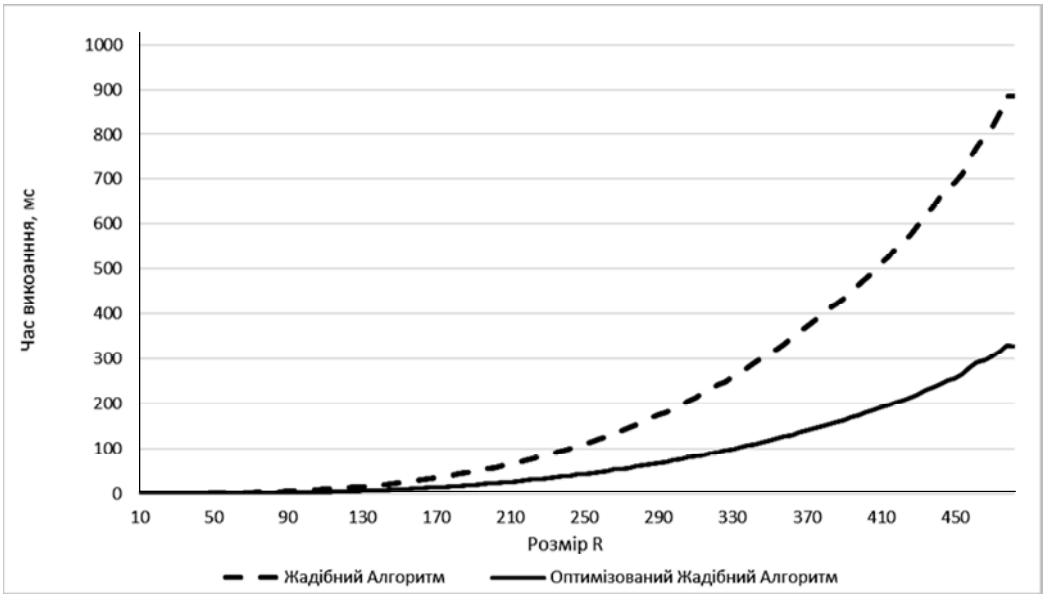

Рис. 6. Залежність часу виконання від розміру R для жадібного і оптимізованого жадібного алгоритму

Експериментальні дослідження підтверджують ефективність оптимізованого алгоритму. Час виконання оптимізованого алгоритму при R=302 в 2.57 разу менший, ніж для звичайного його варіанта. Для R=482 час виконання менший у 2.7 разу. На графіках з рис. 5 і 6 простежуються спільні закономірності: час виконання росте експоненціально, як і кількість операцій; розрив між часом виконання і кількістю операцій для варіацій алгоритмів теж збільшується із зростанням розміру R. Проте для графіка часу виконання ця різниця зростає повільніше – це пояснюється наявністю ієрархії пам'яті в обладнанні, яка згладжує ці залежності. Наявність кеш-пам'яті дозволяє здійснювати швидкий доступ до даних, якщо вони часто використовуються або знаходяться поруч в адресному просторі. У випадку з оптимізованим алгоритмом доступ до даних відбувається стрибкоподібно – на відміну від послідовного, для звичайного алгоритму це призводить до втрати швидкодії. Однак, при великих значеннях R різкі переходи відбуватимуться рідше та вплив на продуктивність знижуватимуться – це видно на графіку на рис. 6.

Висновки

Розглянуто проблеми підвищення продуктивності жадібних алгоритмів у випадку, коли вони використовуються для оптимізації розміщення / маршрутизації компонентів обчислювальних систем.

Запропоновано оптимізований жадібний алгоритм, що має нижчу порівняно з відомими обчислювальну складність. Кількість операцій цього алгоритму дорівнює $\frac{R^3}{8}+\frac{11R^2}{8}-\frac{11R}{4}+1$ проти $\frac{R^2(R-1)}{2}$ для звичайного жадібного алгоритму. Для випадку, коли R = 482, швидкість виконання зростає у 2.7 разу. Зростає швидкість виконання порівняно зі звичайним жадібним алгоритмом разом зі зростанням розміру R, відповідно до графіка на рис. 6.

1. Moore G. E., "No Exponential Is Forever: But 'Forever' Can Be Delayed!" Solid-State Circuits Conference, IEEE International Digest of Technical Papers, Vol. 1, 2003, pp. 20-23. 2. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Глава 16. Жадные алгоритмы // Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. – 2-е изд. – М.: Вильямс, 2005. – 1296 с. – ISBN 5-8459-0857-4. 3. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, Introduction to Algorithms (MIT Press, 2000). 4. Mazur, David R. (2010), Combinatorics: A Guided Tour, Mathematical Association of America, ISBN 978-0-88385-762-5.