УДК 519.161

М. В. Остапчук,
Київський національний університет імені Тараса Шевченка

NP (nondeterministic polynomial) – повні задачі

Розглянуто клас задач, які називають NP-повними. Для задач такого типу не знайдено поліноміальних алгоритмів, проте і не доведено, що таких не існує. Автор описує методи розв'язування цих задач та наводить приклади їх практичного застосування. Як приклад NP-повної задачі розглянуто «Задачу про кліку». Представлено новий алгоритм перерахування всіх максимальних клік у розріджених графах за допомогою алгоритму Епстена, Стреша та проаналізовано його виконання на реальних графах. Цим алгоритмом уперше запропоновано практичне рішення перерахунку всіх максимальних клік великих розріджених графів. Усі інші теоретично швидкі алгоритми для розріджених графів значно повільніші, ніж алгоритм Томіта й інших, на практиці. Проте алгоритмом Томіта й ін. використовується матриця суміжності, яка вимагає занадто багато місця для великих розріджених графів. Наш новий алгоритм відчиняє двері для швидкого аналізу великих розріджених графів, у яких матриця суміжності не впишеться в робочу пам'ять.

Ключові слова: *поліноміальні алгоритми, максимальна кліка, розріджені графи.*

Остапчук М. В. NP (nondeterministic polynomial) – полные задачи.

Рассмотрен класс задач, называемый NP-полными. Для задач данного типа не найдены полиномиальные алгоритмы, однако и не доказано, что таковых не существует. Автор описывает методы решения этих задач и приводит примеры их практического использования. В качестве примера NP-полной задачи рассмотрена «Задача о клике». Представлен новый алгоритм перечисления всех максимальных клик в разреженных графах при помощи алгоритма Эпстена, Стреша и проанализировано его исполнение на реальных графах. Данным алгоритмом впервые предложено практическое решение перечисления всех максимальных клик больших разреженных графов. Все прочие теоретически быстрые алгоритмы для разреженных графов значительно медленнее, чем алгоритм Томита и других, на практике. Но алгоритмом Томита и др. используется матрица смежности, требующая слишком много места для больших разреженных графов. Наш новый алгоритм открывает дверь для быстрого анализа больших разреженных графов, у которых матрица смежности не впишется в рабочую память.

Ключевые слова: *полиномиальные алгоритмы, максимальная клика, разреженные графы.*

Ostapchuk M. V. NP (Nondeterministic Polynomial) – Complete Problems.

The article considers the class of problems called NP-complete. For problems of this type are found polynomial algorithms, but not proved that such algorithms do not exist. The author describes methods for solving these problems, and provides examples of their practical application. As an example of NP-complete problems are considered „clique problem". A new algorithm for listing all maximal clicks in sparse graphs using the algorithm of Epsten, Stresh and analyzed its performance on real graphs. This algorithm first proposed a practical solution to all terms of maximum cliques of large sparse graphs. All other theoretically fast algorithms for sparse graphs is much slower than Tomita's algorithm et al. in practice. However, Tomita's algorithm et al. using the adjacency matrix, which requires too much space for large sparse graphs. Our new algorithm opens the door for fast analysis of large sparse graphs whose adjacency matrix will not fit into working memory.

Key words: *polynomial algorithms, maximal click, sparse graphs.*

Постановка проблеми. Більшість задач, цікавих із практичної точки зору, мають поліноміальні алгоритми рішення. Тобто час роботи алгоритму на вході довжини n становить не більш $O(n^k)$ для деякої константи k (не залежної від довжини входу). Не кожна задача має алгоритм рішення, що задовольняє ці властивості. Деякі завдання взагалі не може бути вирішено ніяким алгоритмом. Класичний приклад такої задачі – «проблема зупинки» (з'ясувати, чи зупиняється ця програма на вході). Бувають також задачі, для яких існує алгоритм розв'язку, але час його роботи не є $O(n^k)$ ні для якого фіксованого числа k, тобто алгоритм працює «довго». Якщо провести формальну межу між «практичними» алгоритмами і такими, що становлять лише теоретичний інтерес, то клас алгоритмів, котрі працюють за поліноміальний час, є розумним першим наближенням.

Виклад основного матеріалу. Розглянемо клас задач, які називають NP-повними. Для них не знайдено поліноміальних алгоритмів, проте і не доведено, що таких не існує. Вивчення NP-повних задач пов'язане з так званим питанням «P = NP». Це питання було поставлено 1971 року та є однією з найскладніших проблем теорії обчислень.

Якщо для деякої задачі вдасться довести її NP-повноту, є підстави вважати її практично нерозв'язною. У цьому випадку раціональніше побудувати наближений алгоритм, ніж продовжувати шукати швидкий алгоритм задачі, який її розв'язує.

NP-повнота та класи P і NP

Неформальний опис

Клас P складається із задач, які розв'язуються протягом поліноміального часу роботи. Точніше кажучи, це задачі, які можна вирішити за час $O(n^k)$, де k – деяка константа, а n – розмір вхідних даних задачі.

Клас NP складається із задач, які піддаються перевірці протягом поліноміального часу. Мається на увазі, що якщо ми якимось чином отримуємо «сертифікат» рішення, то протягом часу, поліноміальним чином залежного від розміру вхідних даних задачі, можна перевірити коректність такого рішення.

Будь-яка задача класу P належить до класу NP, оскільки приналежність задач до цього класу означає, що її рішення можна отримати протягом поліноміального часу, навіть не маючи в розпорядженні сертифіката.

Неформально задача належить до класу NPC (їх називають NP-повними, NP-complete), якщо вона належить класу NP і є такою ж складною, як і будь-яка задача з класу NP.

Якщо будь-яку NP-повну задачу можна вирішити протягом поліноміального часу, то для кожної задачі з класу NP існує алгоритм із поліноміальним часом роботи. Більшість учених, які займаються теорією обчислювальних систем, вважають, що NP-повні задачі дуже важко розв'язати, бо при величезній різноманітності таких задач, котрі вивчалися досі, ні для однієї не знайдено розв'язку у вигляді алгоритму з поліноміальним часом роботи. Тож було б украй дивно, якби всі вони виявилися вирішуваними протягом поліноміального часу.

NP-повні задачі виникають у різних областях: в булевій логіці, теорії графів, арифметиці, при розробці мереж, у теорії множин і розбитті, при зберіганні й пошуку інформації, плануванні обчислювальних процесів, у математичному програмуванні, алгебрі й теорії чисел, при створенні ігор і головоломок, у теорії автоматів і мов, при оптимізації програм, у біології, хімії, фізиці тощо.

Для кожної з мов, представленої на рис. 1, NP-повнота доводиться шляхом приведення її до мови, на яку вказує стрілка, що йде від цієї мови. Кореневою виступає мова Circuit-Sat, NP-повнота якої доводиться.

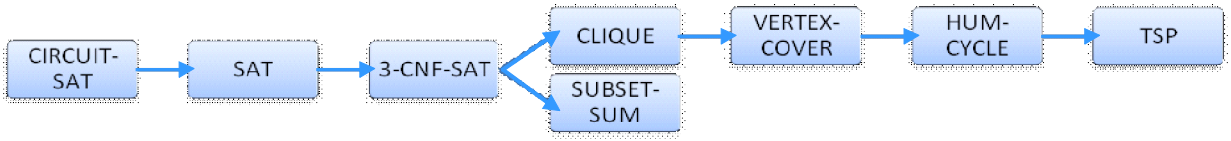


Рисунок 1. Схема доведення NP-повноти для задач. Усі докази зводяться до зведення NP-повної задачі Circuit-Sat до тієї, що розглядається

Як приклад NP-повної задачі розглянемо «Задачу про кліку».

Задача про кліку

Кліка (clique) неорієнтованого графа G = (V, E) – це підмножина V' множини V вершин, кожна пара в якому зв'язана ребром із множиною E. Іншими словами, кліка – це повний підграф графа G. Розмір (size) кліки – це кількість вершин, що містяться в ньому. Задача про кліку (clique problem) – це задача оптимізації, в якій вимагається знайти кліку максимального розміру, що міститься в заданому графі. У відповідній задачі схвалення рішення запитується, чи знаходиться у графі кліка заданого розміру k.

Формальне визначення кліки має вигляд:

CLIQUE = {(G, k) : G – ГРАФ ІЗ КЛІКОЮ РОЗМІРУ k}.

Найпростіший алгоритм визначення того, чи містить граф G = (V, E) з $|V|$ вершинами кліку розміру k, полягає в тому, щоб перерахувати всі k – елементи підмножини множини V і перевірити, чи утворює кожний із них кліку. Час роботи цього алгоритму дорівнює $\Omega(k^2\binom{|V|}{k})$ і є поліноміальним, якщо k – константа. Проте в загальному випадку величина k може сягати значення, близького до $|V|/2$, і в цьому випадку час роботи алгоритму перевищує поліноміальний. Тоді є підстави припускати, що ефективного алгоритму для завдання про кліку не існує.

Теорема. Задача про кліку є NP-повною.

Доведення. Щоб показати, що Clique g $\in$ NP для заданого графа G = (V, E), скористаємося множиною V' $\subset$ V вершин кліки як сертифіката для цього графа. Перевірити, чи є множина вершин V' клікою, можна протягом поліноміального часу, перевіряючи для кожної пари вершин u, $v \in$ V', чи належить ребро (u, v), що сполучає їх, множині E.

Тепер доведемо, що 3-CNF-Sat $\leq_P$ CLIQUE. Звідси випливає, що задача про кліку є NP-складною. Те, що цей результат вдається довести, може здатися дивним, бо на перший погляд логічні формули мало нагадують графи.

Побудова алгоритму зведення починається зі зразка задачі 3-CNF Sat. Нехай $\phi = C_1 \vee C_2 \vee \ldots \vee C_k$ – булева формула з класу 3-CNF із k підвиразами в дужках. Кожен підвираз C_r при r = 1, 2,.., k містить рівно три різні літерали: l_1^r, l_2^r, l_3^r. Сконструюємо такий граф G, щоб формула ϕ виконувалась тоді й тільки тоді, коли граф G містить кліку розміру k.

Це робиться так. Для кожного підвиразу Cr = $(l_1^r V, l_2^r V, l_3^r)$. у формулі ϕ у множині V додається трійка вершин v_1^r, v_2^r, v_3^r. Вершини v_i^r, v_i^s з'єднуються ребром, якщо справедливі два сформульованих нижче твердження:

– вершини v_i^r, v_i^s належать різним трійкам, тобто $r \neq s$;

– літерали, які відповідають цим вершинам, сумісні (consistent), тобто l_i^r не є логічним запереченням l_i^s.

Такий граф легко побудувати на основі формули ϕ протягом поліноміального часу.

Граф G, отриманий у процесі зведення задачі 3-CNF_Sat до задачі CLIQUE, відповідає формулі:

$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \ldots (\neg x_1 \vee x_2 \vee x_3) \ldots (x_1 \vee x_2 \vee x_3)$.

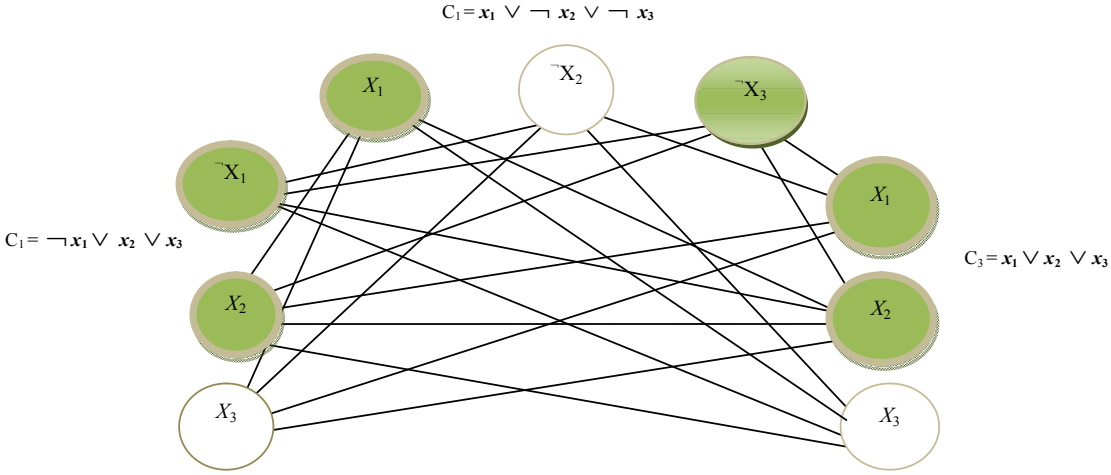


Рисунок 2. Граф, який відповідає формулі $\phi = C_1 \vee C_2 \vee C_3$. Набір $x_1=0$, $x_2=0$, $x_3=1$ виконує C_1 за рахунок $\neg X_2$, а C_2 і C_3 – за рахунок x_3. Відповідні вершини світлі й утворюють клік

Необхідно показати, що це перетворення формули ϕ у граф G є зведенням. По-перше, припустимо, що для формули ϕ є набір, який виконується. Тоді кожний вираз C_r містить не менше одного літерала l_i^r, значення якого дорівнює 1, і кожен такий літерал відповідає вершині v_i^r. У результаті витягання такого «істинного» літерала з кожного підвиразу виходить множина V', що складається з k вершин. Ми стверджуємо, що V' – кліка. Для будь-яких двох вершин v_i^r, v_i^s, що належать V', де $r \neq s$, обидва відповідні літерали і при цьому наборі, який виконується, дорівнюють 1, тому вони не можуть бути запереченнями один одного. Таким чином, відповідно до способу побудови графа G, ребро (v_i^r, v_i^s) належить множині E.

Проведемо зворотні міркування. Припустимо, що граф G містить кліку V' розміру k. Жодне з її ребер не сполучає вершини однієї й тієї ж трійки, тому кліка V' містить рівно по одній вершині кожної трійки. Кожному літералу l_i^r, такому, що v_i^r, які належать V', можна присвоїти значення 1, не побоюючись того, що це значення буде присвоєно як літералу, так і його доповненню, оскільки граф G не містить ребер, які сполучають суперечливі літерали. Кожен підвираз у дужках є виконуваним, тому формула ϕ теж виконувана. (Змінним, яким не відповідає жодна вершина кліки, можна привласнювати довільні значення).

У прикладі, представленому на рис. 2, набір, який виконується для формули ϕ, має вигляд $x_1 = 0$, $x_2 = 0$, $x_3 = 1$. Відповідна кліка розміру $k = 3$ складається з вершин, що відповідають літералу $\neg x_2$ з першого підвиразу в дужках, літералу x_3 з другого виразу в дужках і літералу x_3 з третього виразу в дужках. Оскільки кліка не містить вершин, відповідних літералу x_1 або літералу $\neg x_1$, змінна x_1 у цьому наборі може набувати як значення 0, так і значення 1.

При доведенні теореми довільний випадок задачі 3 – CNF Sat був зведений до випадку задачі Clique, що має певну структуру. Може здатися, що приналежність завдання CLIQUE категорії NP-складних задач була доведена лише для графів, усі вершини яких можна розбити на трійки, причому такі, в котрих відсутні ребра, які з'єднують вершини однієї й тієї ж трійки. Насправді NP-складна задача Clique була показана тільки для цього обмеженого випадку, проте такого доведення достатньо, щоб зробити висновок про NP-складність цього завдання для графа загального вигляду. Бо з наявності алгоритму з поліноміальним часом роботи, що розв'язує задачу CLIQUE із графами загального вигляду, випливає існування такого алгоритму рішення цієї задачі з графами, які мають обмежену структуру.

Але недостатньо звести випадок задачі 3 – CNF_Sat, що має якусь особливу структуру, до випадків задач CLIQUE загального вигляду. Може статися так, що випадок задачі 3 – CNF_Sat, за допомогою якої виконується зведення, виявиться занадто «легким», і до задачі CLIQUE зводиться задача, яка не є NP-складною. Для зведення використовується випадок задачі 3 – CNF_Sat, але не її рішення. Було б помилкою базувати зведення протягом поліноміального часу на знанні того, чи виконується формула ϕ, оскільки не відомо, як отримати цю інформацію протягом поліноміального часу.

Визначення списку всіх максимальних клік великих розріджених графів

Ми реалізуємо новий алгоритм перерахування всіх максимальних клік у розріджених графах за допомогою алгоритму Епстена, Стреша й проаналізуємо його виконання на реальних графах. Наш аналіз показує, що цей алгоритм першим запропонував практичне рішення перерахунку всіх максимальних клік великих розріджених графів. Усі інші теоретично швидкі алгоритми для цих графів значно повільніші, ніж алгоритм Томіта та ін. на практиці. Проте алгоритмом Томіта та ін. використовується матриця суміжності, яка вимагає занадто багато місця для великих розріджених графів. Наш новий алгоритм відчиняє двері для швидкого аналізу великих розріджених графів, у яких матриця суміжності не впишеться в робочу пам'ять.

Процедура пошуку кліки виникає у вирішенні широкого кола важливих прикладних задач. Задача знаходження кліки була вперше розглянута в аналізі соціальних мереж як спосіб пошуку тісно взамодіючих спільнот у них [7]. В області біоінформатики процедура знаходження кліки була використана, щоб знайти закономірності, що часто відбуваються в білкових структурах, аби передбачити структуру білків і їх молекулярних послідовностей та знайти подібність у формах, які можуть вказувати на функціональні зв'язки між білками. Інші програми пошуку кліки включають у себе пошук інформації, комп'ютерного бачення, обчислювальної топології та електронної комерції. Для багатьох прикладних задач ми хочемо знайти не одну велику кліку, а всі максимальні кліки.

Будь-який алгоритм, який вирішує цю проблему, повинен мати експоненціальний час у гіршому випадку, тому що графи можуть містити експоненціальне число клік [9]. Одначе, графи, які зустрічаються у найгіршому випадку, як правило, не зустрічаються на практиці. Швидше за все, типи графів, з якими ми зіткнемося, розріджені [6]. Таким чином, виконання алгоритмів пошуку списку клік полягає в їх здатності належним способом обробляти розріджені вхідні графи. Справді, вже давно відомо, що деякі розріджені сім'ї графів, такі як планарні графи і графи з мінімальною кількістю ациклічних підграфів, містять тільки лінійне число клік, і всі максимальні кліки в цьому графі можуть бути перераховані в лінійний час [3, 4]. Крім того, існує кілька методів, щоб перерахувати всі кліки за поліноміальний час. Пошук клік і їх кількості [11] можна зробити швидше, якщо параметризувати за максимальним ступенем розрідженості.

Багато різних алгоритмів пошуку кліки були реалізовані, як і алгоритм Томіта й ін. [10], базуючись на раніше запропонованому алгоритмі Брона–Кербоша [1], що був визначений через безліч експериментів і на практиці був швидшим на кілька порядків, ніж інші.

Недоліком алгоритму Томіта й ін. є те, що його теоретичний аналіз та реалізація покладається на використання на вході матриці суміжності графа. З цієї причини алгоритм має обмежену застосовність для великих розріджених графів, матриця суміжності якого може не поміститися в оперативній пам'яті. Тому, прагнучи до покращення алгоритму, ми в ідеалі хотіли знайти алгоритм, який конкурує за швидкістю з алгоритмом Томіта й ін., а в результаті мати лінійну вартість зберігання даних.

Останнім часом Мартен Лофле та ін. розробили й опублікували новий алгоритм для включення максимальних клік, зокрема, оптимізований для випадку, коли вхідний граф є розрідженим [5]. Цей новий алгоритм поєднує в собі риси як алгоритму від Томіта й ін. і раніше запропонованого алгоритма Брона–Кербоша, на якому він заснований, та через рекурсивні виклики будує динамічний граф як структуру даних, що представляє суміжності між вершинами, які залишаються актуальними протягом кожного виклику.

За допомогою параметризованої складності в плані виродження вхідного графа (міра його розрідженості) час його роботи є майже оптимальним з точки зору найгіршого числа клік, із яким граф може бути. Однак попередні роботи авторів на чолі з Мартеном Лофле не включають рішення або експериментальні результати, все-таки показують, що алгоритм добре працює і в теорії, і на практиці.

Наші результати

Ми реалізуємо алгоритм Епстена, Лофле, Стреша для перерахування всіх максимальних клік у розріджених графах [5]. Реалізуємо модифіковану версію Томіта й ін. – алгоритм, який використовує списки суміжності замість матриці суміжності.

Наші результати показують, що при великих розріджених графах новий алгоритм є швидшим, ніж Томіта й ін., іноді у дуже багато разів.

Для нерозріджених графів новий алгоритм іноді повільніший, ніж алгоритм Томіта й ін., але залишається в межах невеликої постійної величини.

Попередні приготування

Ми працюємо з неорієнтованим графом G = (V, E) з n вершинами і m ребрами. Для вершини v, нехай G (v) множина сусідніх вершин $\{w : (v;w) \in E\}$, аналогічно для підмножини $W \subset V$, $G(W) = \cap_{w \in W} G(W)$ множина буде спільним сусідом для усіх вершин з W. Виродження графа G – це таке найменше число d, що кожен підграф графа G містить вершину степеня не вищу за d.

Алгоритм Брона–Кербоша зі стратегією вибору провідного елемента Томіта й ін.

proc Tomita(P, R, X)

1: if $P \cup X = \varnothing$ then

2: report R as a maximal clique

3: end if

4: choose a pivot $u \in P \cup X$ to maximize $|P \cap G(u)|$

5: for each vertex $v \in P \backslash G(u)$ do

6: Tomita($P \cap G(v)$, $R \cup \{v\}$, $X \cap G(v)$)

7: $P \leftarrow P \backslash \{v\}$

8: $X \leftarrow X \cup \{v\}$

9: end for

Кожен граф із виродженням *d* має впорядковане вироджене, лінійно впорядковані вершини такі, що кожна вершина має не більше *d* сусідів далі, ніж вона є в упорядкуванні. Вироджене вхідного графа і впорядковане вироджене графа може бути обчислене за лінійний час.

Алгоритм Томіта й ін.

Алгоритм Томіта й ін. [10] є реалізацією алгоритму Брона–Кербоша [8], що використовує евристику з вибором провідного елемента [8, 2]. Алгоритм Брона–Кербоша – це простий рекурсивний алгоритм, який оперує з трьома наборами вершин: часткова кліка R, набір кандидатів для розширення кліки P і набір заборонених вершини X. У кожному рекурсивному виклику вершина *v* з P додається до часткової кліки R, множина кандидатів розширення та множина заборонених вершин обмежені включенням тільки сусідів *v*.

Якщо $P \cup X$ стає порожня, то алгоритм повідомляє, що R – у максимальній кліці, але якщо P стає порожня, а X не пуста, то алгоритм повертається, не визначивши кліки. В основній версії алгоритму виконується $|P|$ рекурсивних викликів, по одному для кожної вершини з P. Евристика з вибором провідного елементу зменшує кількість рекурсивних викликів, вибираючи вершину u в $P \cup X$, яку назвемо провідною вершиною.

Усі максимальні кліки не повиннні містити сусідів вершини u (вважатимемо u як не-сусіда), і, отже, рекурсивні виклики можуть бути обмежені до перетину P з не-сусідами. Алгоритм Томіта й ін. вибирає провідну вершину так, що вершина u має максимальну кількість сусідів у P, і тому мінімальне число не-сусідів серед усіх можливих провідних вершин. Розрахунок провідних вершин і множин для рекурсивних викликів може бути зроблений з $O(3^{n/3})$ для перерахування всіх максимальних клік [10].

Алгоритм Епстена, Лофле і Стреша

Епстен, Лофле і Стреш [5] дають інший варіант алгоритму Брона–Кербоша, який для гіршого випадку отримує близьку до оптимальної оцінку часу для графів з невеликим виродженням.

Він спочатку обчислює порядок виродження графа, зовнішній виклик рекурсивного алгоритму вибирає вершини *v* так само, як вони будуть використовуватися в кожному рекурсивному виклику, в такому ж порядку та без вибору провідного елементу. Тоді для кожної впорядкованої вершини *v* здійснюється виклик алгоритму від Томіта й ін. [10] для обчислення всіх клік, що містять *v* і сусідів *v*, що стоять далі, у той час не враховуючи сусідів *v*, що стоять раніше. Порядок виродження обмежується множиною P, і рекурсивних викликів буде не більше, ніж *d*, виродження графа.

Проста стратегія для визначення провідних елементів при кожному зверненні до алгоритму Томіта і співавт., що використовуються як підпрограма в рамках цього алгоритму, організовує цикл за всіма можливими провідними елементами в $P \cup X$. Для кожного з них буде виконано цикл за його сусідами, що стоять далі, у порядку виродження графа, щоб визначити, скільки з них знаходяться в P. Та ж сама стратегія може бути використана для виконання перетинів сусідніх елементів, що вимагають рекурсивних викликів. Із провідною вершиною і набором перетинів, реалізованих таким чином, алгоритм працює за час $O(d^2 n 3^{d/3})$, коефіцієнт *d* більший, ніж у найгіршому випадку розмір вихідного результату, що дорівнює $O(d(n-d)3^{n/3})$.

Однак Епстен та співавт. забезпечили уточнення цього алгоритму, що зберігає на кожному рівні рекурсії підграф G з вершинами в $P \cup X$ і ребра, які мають хоча б один кінець у P. Використовуючи цей підграф, зменшуємо час обчислень провідних вершин до $|P|(|X|+|P|)$, і перетини з сусідніми для кожного рекурсивного виклику займають час $|P|^2(|X|+|P|)$, що зменшує загальний час на $O(dn3^{d/3})$. Цей час роботи відповідає найгіршим за розміром вихідним даним задачі, коли $d \le n - \Omega(n)$.

Як описано Епстеном і співавт., зберігання підграфів на кожному рівні рекурсії вимагає $O(dm)$ простору. Але ми покажемо, що можна досягти того ж оптимального часу роботи $O(n+m)$.

Алгоритм Томіта й ін. Списки суміжності

Алгоритм Томіта й ін. [10] працює на графах малого та середнього розмірів через використання матриці суміжності. Для того, щоб виконати алгоритм на великих графах, ми реалізовуємо простий варіант алгоритму, який зберігає вхідні дані граф у списку суміжності й виконує обчислення провідної вершини, перебираючи всі вершини $P \cup X$, і тестування всіх сусідів на наявність в P. Коли вершина v додається до рекурсивного виклику, ми можемо виконувати перетин сусідів *r* з P і X перебирає її сусідів таким же чином.

Нехай Δ буде максимальним степенем даного вхідного графа, тоді обчислення провідних вершин займає багато часу: $(O\Delta(|X|+|P|))$. Крім того, приготування підмножин для всіх рекурсивних викликів вимагає часу $O(|P|\Delta)$. Розгляд цих фактів в аналіз Томіта й ін. дає нам час $O(\Delta(n-\Delta)3^{\Delta/3})$.

Δ може бути значно більше, ніж виродження, тому теоретичні межі часу виконання цього алгоритму не такі ж хороші, як Томіта й ін. або Епстена та ін. Одначе, простота цього алгоритму робить його конкурентоспроможним з іншими у багатьох випадках вирішення цієї задачі.

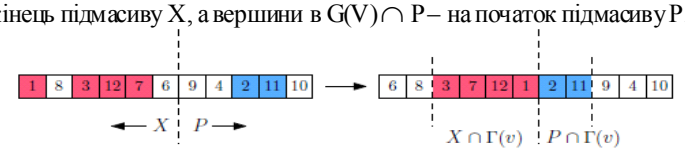
Реалізація та виконання

Ми реалізували алгоритм Томіта й співав., використовуючи подання графа, представленого списком суміжності, користуючись тим, що вершини мають декілька сусідів, у впорядкованому виродженні реалізується динамічна структура графа даних, яка використовується тільки O ($m + n$) додаткового загального простору.

Ми маємо множини вершин P й X у єдиному масиві, який передається між рекурсивними викликами.

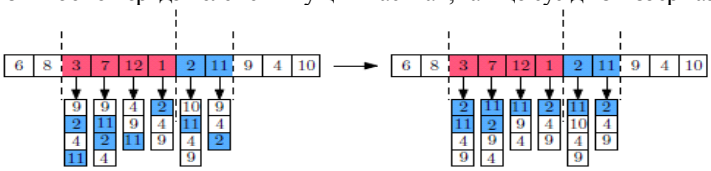
Спочатку масив містить елементи X, за яким слідують елементи P. Ми застосуємо зворотний пошук у таблиці, так що можемо знайти індекс вершини у фіксований час. За допомогою цієї таблиці пошуку зможемо сказати, чи вершина знаходиться в P, чи X за фіксований час, перевіряючи її індекс у відповідному підмасиві.

Коли вершина v додається до R при підготовці до рекурсивного виклику, ми перепорядковуємо масив. Вершини з $G(V) \cap X$ переміщаються в кінець підмасиву X, а вершини в $G(V) \cap P$ – на початок підмасиву P.



Потім робимо рекурсивний виклик підмасиву, що містить вершини $G(V) \cap (X \cup P)$. Після рекурсивного виклику перекидаємо v до X шляхом заміни її на початок підмасиву P і переміщаємо межі, так що v належить підмасиву X. Звичайно, переміщення вершин між множинами вплине на P і X у вищих рекурсивних викликах.

Таким чином, у даному рекурсивному виклику ми оновлюємо список вершин, які перемістилися з P в X, і переміщаємо ці вершини до P, коли виклик закінчується. Розрахунок провідних вершин структури даних зберігається у вигляді набору масивів, по одному для кожної потенційної провідної вершини u в $P \cup X$, що містить сусідів u в P. Кожного разу при зміні P ми змінюємо порядок елементів у цих масивах, так що сусіди з P зберігається першими.



Обрахувати провідні вершини так само просто, як ітерації по кожному масиву, поки ми зустрічаємо сусіда не з P.

Ця процедура перепризначення дозволяє обслуговувати одним набором масивів усі рекурсивні виклики, які потребують спільного лінійного простору. Аби зробити нову копію цієї структури даних для кожного рекурсивного виклику, потрібний простір O (dm).

Вивчення теорії складності та класифікація завдань з точки зору NP-повноти має практичний зміст, бо набагато розумніше й ефективніше знайти доведення того, що дана задача належить до класу NP-повних, і тому зайнятися пошуком досить точних наближених алгоритмів, ніж безрезультатно витрачати час на знаходження поліноміальних алгоритмів її рішення. Ясно, що саме NP-повні задачі відіграють тут центральну роль. Річ у тому, що поліноміальний час є хоч і першим, але досить хорошим наближенням поняття «практична вирішуваність задачі».

Висновки. Ми показали, що алгоритм Епстена, Лофре, Стреша це практичний алгоритм для застосування на великих розріджених графах. Перевагою цього алгоритму є те, що він вимагає тільки лінійний простір для зберігання графа і всіх структур даних. Він не потребує матриці суміжності, яка може не поміститися в пам'ять. Його найближчий конкурент із цього погляду – алгоритм Томіта й ін. Таким чином, алгоритм Епстена та співавт. – це швидкий і надійний вибір для виявлення максимальних клік, особливо коли вхідний граф великий і розріджений.

Література

1. Bron C. Algorithm 457: finding all cliques of an undirected graph / C. Bron, J. Kerbosch // Communication of ACM. – Vol. 16(9). – 1973. – P. 575–577.

2. Cazals F. A note on the problem of reporting maximal cliques / F. Cazals, C. Karande // Theoretical Computer Science. – Vol. 407(1–3). – 2008. – P. 564–568.

3. Chiba N. Arboricity and subgraph listing algorithms / N. Chiba, T. Nishizeki // SIAM Journal on Computing. – Vol. 14(1). – 1985. – P. 210–223.

4. Chrobak M. Planar orientations with low out-degree and compaction of adjacency matrices / M. Chrobak, D. Eppstein // Theoretical Computer Science. – Vol. 86(2). – 1991. – P. 243–266 ; 2002. – P. 157–206.

5. Eppstein D. Listing all maximal cliques in sparse graphs in nearoptimal time / D. Eppstein, M. Loffer, D. Strash. In: Cheong O., Chwa K. Y., Park K. (eds.) // ISAAC, Lecture Notes in Computer Science. – Vol. 6506. – 2010. – P. 403–414.

6. Goel G. Bounded arboricity to determine the local structure of sparse graphs / G. Goel, J. Gustedt. In: Fomin F. (ed.) WG, Lecture Notes in Computer Science. – Vol. 4271. – 2006. – P. 159–167.

7. Harary F. A procedure for clique detection using the group matrix / F. Harary, I. C. Ross // Sociometry. – Vol. 20(3). – 1957. – P. 205–215.

8. Koch I. Enumerating all connected maximal common subgraphs in two graphs / I. Koch // Theoretical Computer Science. – Vol. 250(1–2). – 2001. – P. 1–30.

9. Moon J. W. On cliques in graphs / J. W. Moon, L. Moser // Israel Journal of Mathematics. – Vol. 3(1). – 1965. – P. 23–28.

10. Tomita E. The worst-case time complexity for generating all maximal cliques and computational experiments / E. Tomita, A. Tanaka, H. Takahashi // Theoretical Computer Science. – Vol. 363(1). – 2006. – P. 28–42.

11. Tsukiyama S. A new algorithm for generating all the maximal independent sets / S. Tsukiyama, M. Ide, H. Ariyoshi, I. Shirakawa // SIAM Journal on Computing. – Vol. 6(3). – 1977. – P. 505–517.

12. Ахо А. Построение и анализ вычислительных алгоритмов / А. Ахо, Д. Хопкрофт, Д. Ульман. – М. : Мир, 1979. – С. 420–427.

13. Кнут Д. Искусство программирования / Д. Кнут // Основные алгоритмы // The Art of Computer Programming. – Vol. 1. Fundamental Algorithms. – 3-е изд. – М. : Вильямс, 2006. – С. 720.